Improvements to fMPE for discriminative training of features

Daniel Povey

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

Abstract

fMPE is a previously introduced form of discriminative training, in which offsets to the features are obtained by training a projection from a high-dimensional feature space based on posteriors of Gaussians. This paper presents recent improvements to fMPE, including improved high-dimensional features which are easier to compute, and improvements to the training procedure. Other issues investigated include cross-testing of fMPE transforms (i.e. using acoustic models other than those with which the fMPE was trained) and the best way to train the Gaussians used to obtain the vector of posteriors.

1. Introduction

fMPE, introduced recently [1] is a discriminative training technique that uses the Minimum Phone Error (MPE) discriminative criterion [3] to train a feature-level transformation. fMPE was one ingredient of the system with which IBM won the English conversational telephone speech (CTS) component of the recent EARS evaluation [2].

Section 2 briefly describes fMPE and the training procedures used. Section 3 describes some improvements to the high-dimensional features used for fMPE. Section 4 describes some improvements to the gradient descent method used to train the parameters. Section 5 describes a layer-based framework used to implement fMPE that makes the approach easy to extend. Section 6 investigates cross training and joint training of fMPE transforms between different acoustic models. Section 7 discusses the training of feature "variances" and the question of how to obtain the Gaussians used to get the posteriors used in fMPE; and Section 8 gives conclusions.

2. Introduction to fMPE

The original formulation of fMPE presented in [1] was based on the feature transformation:

$$\mathbf{y}_t = \mathbf{x}_t + \mathbf{M}\mathbf{h}_t, \quad (1)$$

where $\mathbf{x}_t$ is the old feature vector, and the new feature vector $\mathbf{y}_t$ equals the old feature vector plus a high dimensional time-specific vector $\mathbf{h}_t$ times a matrix $\mathbf{M}$. The vector $\mathbf{h}_t$ is a function of $\mathbf{x}_t$ and possibly adjacent frames as well, and is derived from $\mathbf{x}_t$ by forming a vector of posteriors of Gaussians. The large matrix $\mathbf{M}$ is trained using the MPE objective function from a zero start.

It has been previously described [1] how the MPE objective function is differentiated to obtain the gradient $\partial\mathcal{F}/\partial\mathbf{y}_t$, i.e. the gradient w.r.t the feature vector on each time. As with normal MPE training, this process includes the generation of lattices by decoding the training data with a weak language model, and performing forward-backward like algorithms on the lattices. The gradient w.r.t. the features contains two terms; one reflects changes in likelihood for different transcriptions of the training data assuming the Gaussian parameters in the system stay fixed, and the other reflects the fact that the training data affects the Gaussian means and variances, which in turn affect the MPE objective function indirectly. This second ("indirect") differential is necessary because the means and variances are to be trained with ML on the features, whereas the features are discriminatively trained. If fMPE is to be combined with discriminative training of the HMM parameters, MPE is performed after fMPE. This has been empirically found to be the best approach.

The matrix $\mathbf{M}$ is trained using a modified form of gradient descent. The two main differences with normal gradient descent are that, first, the positive and negative contributions to each parameter's differentials are accumulated separately and the inverse of the sum of their absolute values is used as a factor in the learning rate (this means that the learning rate is proportional to the ratio between the positive and negative parts, not their absolute values). Also a factor is included in the learning rate which reflects the average standard deviation of the feature dimension we are adding to. A more minor point is that a form of backoff is used which causes differentials that have very low effective counts (i.e. non-robust differentials) to approach zero.

3. Improvements to features

3.1. Original features

The original fMPE features used were based on posteriors of Gaussians with frame splicing. A set of, say, 100,000 Gaussians was obtained by clustering the set of Gaussians in the HMM set to 100,000 clusters using a likelihood based clustering procedure and using the 100,000 cluster centers (which are themselves Gaussians). These 100,000 Gaussians are evaluated on each frame (this can be done efficiently by further clustering them to, say, 2000 clusters and only evaluating those corresponding to the most likely cluster centers). On each frame the vector of Gaussian posteriors given the feature vector $\mathbf{x}_t$ is taken as the basic feature vector $\mathbf{h}_t$. These are spliced together with adjacent frames and averages of adjacent frames: for instance the central frame, frame 1 (one to the right), frames 2 and 3; frames 4 and 5; frames 6,7 and 8; and the same to the left, 9 contexts in total so the spliced feature vector size would be 900,000 in this case.

3.2. Offset features

An improved feature vector is now being used, which gives better error rates and also saves time on Gaussian evaluation. The idea is to use far fewer Gaussians (e.g. 1000), and supplement the posterior of the n'th Gaussian (γ_n) with the offset of the

		Iteration				
	#Gauss	MLE	1	2	3	4
Baseline	64000	25.0	23.4	22.9	22.7	22.5
Offset	1000	25.0	23.2	22.6	22.3	22.1

Table 1: Baseline (posterior only) features vs. posteriors with feature offsets. CTS (Fisher+Switchboard etc), 250h training, VTLN+fMLLR, testing on RT'03

observed feature vector from the mean of the n'th Gaussian, scaled by the n'th posterior. In order to obtain features with a normalized scale, we also divide by the standard deviation. In addition the posterior itself is scaled up (to give it more weight against the larger number of offsets), so the final vector looks like:

$$[5.0\gamma_1, \gamma_1(x_t(1)-\mu_1(1))/\sigma_1(1), \gamma_1(x_t(2)-\mu_1(2))/\sigma_1(2), \ldots,$$
$$5.0\gamma_2, \gamma_2(x_t(1)-\mu_2(1))/\sigma_2(1), \gamma_2(x_t(2)-\mu_2(2))/\sigma_2(2), \ldots].$$

The dimension of the final feature vector would be $N(d+1)$ if N is the number of Gaussians and d is the dimension of the input features.

Offset features were originally developed during experimentation with a call-center speaker independent setup, where they gave around 1% absolute improvement from a 30% baseline. Most subsequent experiments have used offset features. In order to verify that they generally give improvements (and not only in the original setup), an experiment was performed on a speaker adapted 250h Switchboard training setup. Results are given in Table 1. The offset features give about 0.4% absolute improvement in WER. Both systems include training of the context expansion, as described below.

3.3. Reorganized context calculation

With the offset features in place, more of the elements of $\mathbf{h}_t$ are nonzero, and when combined with context expansion this can cause efficiency problems because the number of nonzero elements in $\mathbf{h}_t$ can number in the hundreds. To solve this the fMPE calculation has been reorganized.

The raw, un-spliced form of the vector $\mathbf{h}_t$ is projected to, a dimension of, say, $9d$ (where d is the dimension of the features) by a matrix $\mathbf{M}_1$ with $9d$ rows, giving us $\mathbf{v}_t = \mathbf{M}_1\mathbf{h}_t$. The vectors $\mathbf{v}_t$ are then projected down to dimension d by a transformation that performs averaging and summing across time of the contexts of each dimension, with only 9 of the elements of $\mathbf{v}_t$ affecting each dimension in the output. This can be expressed in matrix notation as follows.

Combine the vectors $\mathbf{v}_t$ across a window of time to make a matrix:

$$\mathbf{V}_t = [\mathbf{v}_{t-f}\mathbf{v}_{t-f+1}\ldots\mathbf{v}_{t+f}],$$

and obtain from it a $9d$ dimensional vector $\mathbf{w}_t = \mathbf{M}_2 * \mathbf{V}_t$, where $*$ is defined as a matrix operator that "multiplies" two matrices of identical dimension by taking the dot product of corresponding rows and gives the result as a column vector. $\mathbf{M}_2$ is a $9d \times (2f+1)$ matrix (where f is the number of frames of context on each side), which is initialized to have the same effect as the context expansion use for the baseline features (described in Section 3.1). That means the first d rows would have a single 1.0 in the center of the row, the next d rows would have a 1.0 in the element to the right of the center, etc. Finally, $\mathbf{w}_t$ is collapsed to dimension d by breaking its elements up into 9 blocks and adding the blocks together. The result is added to the original feature vector $\mathbf{x}_t$ to form the final vector $\mathbf{y}_t$.

	Iteration					
Context	MLE	1	2	3	4	5
Fixed	44.8	40.5	39.8	38.8	38.3	38.0
Trained	44.8	40.5	39.2	38.1	37.9	37.5

Table 2: Context expansion left fixed vs. training the expansion. IBM call-center data.

The calculation described above is exactly equivalent to the original context expansion but more efficient where $\mathbf{h}_t$ has many nonzero elements. It also allows us to train the context expansion $\mathbf{M}_2$. Gradients are calculated by back propagation through the calculation. The matrix $\mathbf{M}_2$ is trained in the same way as $\mathbf{M}_1$, except that the learning rate rule lacks the term σ_i which compensates for different dimension having different variance. Note that the context expansion must be trained with held out data (i.e. different data from that used to train the main matrix $\mathbf{M}_1$). This is because if it is trained on the same data, the context expansion layer attempts to maximize data learning by simply scaling up the fMPE contribution to the features. For experiments reported here, one out of every ten files is held out to train the context expansion.

Table 2 shows the effect on WER of training the context vs. leaving it fixed, on call-center data. Both use mean-offset features with 1000 Gaussians. There seems to be a WER improvement of about 0.5% from training the context.

4. Improvements to learning rules

4.1. Setting learning rates

The learning rate for matrix elements $\mathbf{M}_{ij}$ is controlled by a constant E which is the inverse of the learning rate. Originally this was set by hand based on a knowledge of the features used. Since this requires special knowledge, a different rule is now used. On the first iteration of updating a matrix, E is now set to the value that, based on the present gradient, would give a specified improvement in the MPE criterion. The calculated E value is also used for subsequent iterations. For instance, for the main matrix $\mathbf{M}_1$ a target improvement of 0.06 is typically used, or a smaller value (say 0.02) for tasks with very low error rates such as digits. For the context-training matrix $\mathbf{M}_2$, around 1/10 the target improvement of the main matrix is used. Note that $\mathbf{M}_2$ is only trained from the second iteration, since the differential would be zero on the first iteration.

4.2. Smooth update

It was found on certian occasions that an instability developed in the parameters of certain rows of the matrix $\mathbf{M}_2$ that trains the context expansion. A technique was developed to fix this, which also improves the general predictability and stability of the fMPE training process. This "smooth update" technique is a post-processing stage to be used with existing update rule (a form of gradient descent in this case). The intuition of the smooth update rule is that if too many parameters are changing sign *and* making a larger step size than the last iteration, the learning rate is probably too fast. This is only applicable from the second iteration.

The "smooth update" procedure is as follows, described for iteration n. First, define a meaningful set of sets of parameters (based on knowledge of the task), and list them in decreasing order of size. This set of sets might be, for instance: all parameters; all matrix columns; all sets of matrix rows modulo 39; all

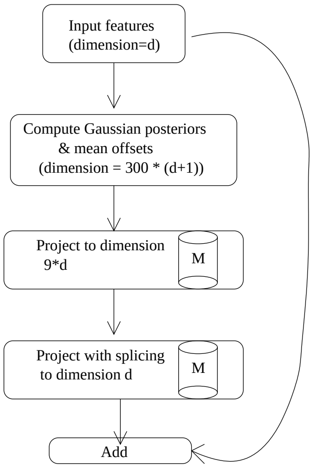


Figure 1: Outline of layer-based fMPE setup with offset features & context training

blocks of 39 matrix rows; all matrix rows. Then, for each set, if more than 10% of the parameters on the n'th iteration are on the opposite side of the value on the $n-2$'th iteration from the value on the $n-1$'th iteration, move the updated parameters (iteration n) towards those on the $n-1$'th iteration (i.e. decrease the learning rate) until this is no longer the case. If the i'th parameter on the n'th iteration is $x_i(n)$, the condition would be: $(x_i(n) - x_i(n-2))(x_i(n-1) - x_i(n-2)) < 0$ for no more than 10% of each set of parameters.

Note that it may be useful to plot graphs of the number of sign changes, and the predicted criterion improvement, for all these sets of parameters.

The main benefit of the "smooth update" method is to provide robustness against occasional divergence. Results are not presented in this paper to compare this method against a baseline, as it would be necessary to test on more than one setup and to work out the best learning rates for both techniques. All results presented here use the smooth update method.

5. Layer code

With the reorganized context calculation in place, it becomes more difficult to sustain an ad hoc approach to programming fMPE. A more flexible framework was developed which allows various operations on speech data to be performed in a normalized way. It involves the concept of a "layer set", which is a sequence of "layer" modules specified by a configuration file (a configuration file contains a set of name-value pairs). Figure 1 shows the typical fMPE setup used currently. This corresponds to a configuration file as follows:

```
layers=layer0+layer1+layer2+layer3+layer4

layer0.type=read
layer0.feature-base=/somedisk/somedir@40_dim_lda_feats # Location of feature files.

# Posterior calculation.
layer1.type=xpost
layer1.post-scale=5.0 # Scale on posterior
layer1.prots=/somedisk/somedir/gaussians.1024_dim
layer1.max-leaves-write=2 # Pruning
layer1.write-thresh=1.0   # Pruning
layer1.has-diff=false # Do not accumulate differentials
                      # w.r.t parameters in this layer.

# Project to a dimension of 40 x 9, for 9 contexts.
layer2.type=project
layer2.input1=layer1
layer2.dim-out=360
# sqdiff relates to how we calculate counts per
# parameter for smoothing low-count diffs:
layer2.use-sqdiff=true  # Re calculating counts for smoothing
layer2.prev-matrix=/somedisk/somedir/transform.iter1.layer2
layer2.matrix-name=/somedisk/somedir/transform.iter2.layer2
layer2.output-matrix-fn=/somedisk/somedir/transform.iter3.layer2
# for per-dim variance:
layer2.ref-prots=/somedisk/somedir/some_gaussians
layer2.var-type=prots-var
layer2.suggested-impr=0.06  # MPE crit impr on first iteration
layer2.max-sign-changes=0.1 #  10% sign changes max
layer2.smoothupdate-sets=cols:rowblk,41:rowmod,41:rows:count
layer2.tau=100  # Smoothing of low-count differentials.
layer2.has-diff=true
layer2.accept-modulo=10:1,2,3,4,5,6,7,8,9 # Accept 9/10 files
                         # (relates to holding out data).

#Context expansion, done as collapsing of larger features
layer3.type=collapsefeat
layer3.accept-modulo=10:0 # Accept 1/10 files (held out)
layer3.prev-matrix=/somedisk/somedir/transform.iter1.layer3
layer3.matrix-name=/somedisk/somedir/transform.iter2.layer3
layer3.output-matrix-fn=/somedisk/somedir/transform.iter3.layer3
# Specification of initial matrix:
layer3.matrix-string=0,1.0:-1,1.0:1,1.0:-2,0.5;-3,0.5:
2,0.5;3,0.5:-4,0.5;-5,0.5:4,0.5;5,0.5:
-6,0.333;-7,0.333;-8,0.333:6,0.333;7,0.333;8,0.333
layer3.has-diff=true
layer3.start-frame=-40  # Extent of context (f=40)
layer3.end-frame=40
layer3.suggested-impr=0.007
layer3.max-sign-changes=0.1
layer3.tau=100  # Smoothing of low-count differentials.

layer4.type=add
layer4.has-diff=true
layer4.input1=layer0
layer4.input2=layer3

#where to put differentials for each layer.
diff-base=/somedisk/somedir/accs.%d.out
diff-added=/somedisk/somedir/accs.added.out
```

Each layer has a variable "type", which instantiates the layer object with a particular class; most of the rest of the variable names are specific to the type of layer. Each layer reads from the previous layer by default. Note that the layers with high-dimensional sparse output (e.g. `xpostlayer` in the above example) use a sparse representation for their output. This code structure is useful not only for fMPE, but for general feature processing as well.

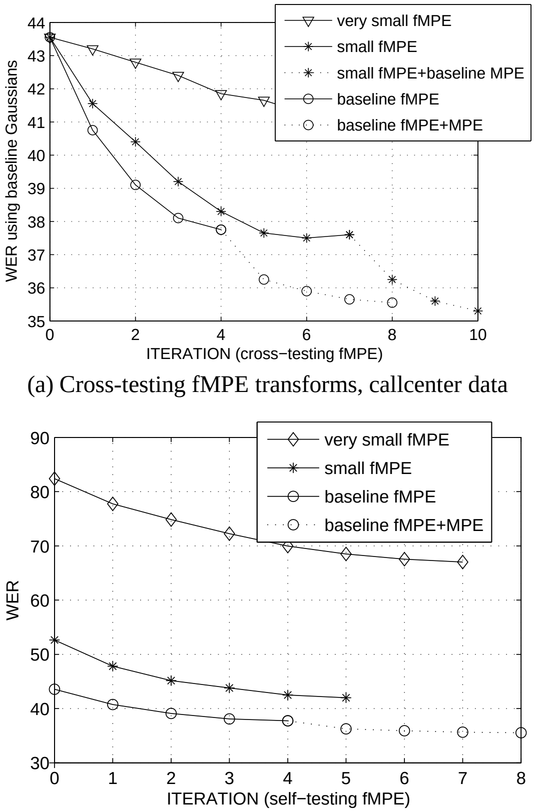
6. Joint and cross training of fMPE

Since the fMPE transformation is trained jointly with the Gaussians in the system, an interesting question is: to what extent are the trained parameters tied to the particular HMM set used? Are they learning something specific, or something general?

Some experiments were performed in which a smaller HMM set was generated from an existing system. A system trained and tested on call-center data, with no speaker adaptation, was the baseline. Two systems were generated from it by reducing the number of mixture components in the Gaussians (no retraining was done after the mix-down procedure). A "small" system had 1/4 the number of Gaussians per state of the original system, and a "very small" system had 1 Gaussian per state, about 4% as many Gaussians as the original system.

The smaller systems were both used to generate lattices and fMPE training was performed on them. The baseline system was retrained on the fMPE features on each iteration of training the two other systems, and testing was performed using both the smaller and baseline systems. For the "small" system, model-space MPE training using the baseline system's Gaussians was continued.

Figure 2(a) shows the result of testing using the baseline size of HMM trained and tested using fMPE trained with the "small" and "very small" systems. With the fMPE features from the "small" system the results are as good or better than self-trained fMPE, but there is much less improvement from the "very small" system. Figure 2(b) shows normal fMPE training and testing using all three systems, to confirm that training is proceeding as it should. The conclusion from this experiment seem to be that the fMPE transform is not too closely tied to the Gaussians of the system, as shown by the fact that the fMPE transform tranined with the "small" system gave good results when tested with the baseline system.



(a) Cross-testing fMPE transforms, callcenter data

(b) Self-testing fMPE transforms, callcenter data

Figure 2: Self and cross-testing of fMPE transforms

Another experiment (not shown) was performed in which fMPE training was done jointly using the Gaussians of the baseline and "small" system, i.e. sharing the fMPE parameters. This gave the best result of all, about 0.5% absolute better than the baseline after both fMPE and MPE training.

7. Other issues investigated

7.1. Training feature "variances"

Some improvement was obtained by, in addition to training the features, training "variances" on the features, i.e. a quantity which is added to all $(x - \mu)^2$ quantities in both training and testing. This involves a straightforward extension to the fMPE formulae (basically, differentiating with respect to these variances). This has been tested in two setups, one in which a system was trained on broadcast news and tested on the broadcast news eval97 test set (unadapted); the improvement after fMPE alone was from 20.1% to 19.3%. On the TC-Star task (European parliamentary speeches) after fMPE and MPE, on features that included VTLN and fMLLR the WER improvement on evaluation data was around 0.3% absolute at the 15% level. However this was not used in the final system as it would have involved changes in the decoding setup. The variances are trained in a similar way to the feature offsets except that variance "offsets" are added to a zero "baseline" variance, and to ensure positive variances, the variance offsets are all added to a constant value and put through a sigmoid function in such a way that the variances are all 0.2 times the average variance of each dimension's features at the start of training, and are limited by the sigmoid to be between 0 and 2.0 times the average variance. It is not clear that the extra computation that this approach requires during Gaussian computation at test time is worth the benefits it gives. Note that a related technique has been tried [1] which uses the MPE objective function to train a global scale on variances; this does not require as much test-time computation.

7.2. Obtaining Gaussians

All fMPE approaches tried to date require first obtaining an arbitrary set of Gaussians which are used to obtain Gaussian posteriors. These Gaussians have generally been obtained by a likelihood based clustering of Gaussians in a trained HMM set down to the number of Gaussians required. A question remained as to whether this was the best approach. In a related technique [4, 5], a set of Gaussians is trained as a general GMM on the speech data. Some experiments were performed on the same call-center setup for which experiments were reported in Figure 2. First, Gaussians were trained for several iterations on speech data starting from the initial clustered Gaussians, both with and without tied variances. Both approaches gave an approximately 0.3% (absolute) worse performance compared to the clustered baseline after three iterations of fMPE, around the 38% level. In another experiment, Gaussians were trained starting from randomly chosen training data points and trained using a common variance for several iterations, being given different variances on the last iteration. This produced around a 0.3-0.5% degradation compared to the baseline clustered Gaussians. Note that fMPE in this setup is giving around 7% absolute improvement so these differences are not too large; they may well be due to something as simple as different coverage of silence versus non-silence.

8. Conclusions

This paper has described recent improvements in the fMPE training setups and has presented results on some representative tasks. The main improvements include new features (which augment the posteriors with offsets from the means of the Gaussians), and a more efficient, trainable way of doing frame splicing of the high-dimensional features for acoustic context. Experiments have been performed with cross-training of fMPE, showing that the fMPE transform does not seem to be too closely tied to the HMM it was trained with and that it can be used with other HMMs. In addition other issues relevant to fMPE training were investigated, such as the manner in which the Gaussians are obtained (it seems not to matter too much).

9. References

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

[2] H. Soltau, B. Kingsbury, L. Mangu, D. Povey, G. Saon & G. Zweig, "The IBM 2004 Conversational Telephony System for Rich Transcription," *ICASSP*, 2005.

[3] D. Povey and P. C. Woodland, "Minimum Phone Error and I-smoothing for Improved Discriminative Training," *ICASSP*, 2002.

[4] J. Droppo, L. Deng & A. Acero, "Evaluation of the SPLICE algorithm on the Aurora2 Database", *Eurospeech*, 2001.

[5] J. Wu & Q. Huo, "An Environment Compensated Minimum Classification Error Training Approach and its Evaluation on Aurora2 Database," *ICSLP*, 2002.

[1] Personal communication from K.C. Sim & M.J.F. Gales.