

MMIE Training for Large Vocabulary Continuous Speech Recognition

Yves Normandin, Roxane Lacouture and Régis Cardin

Centre de recherche informatique de Montréal (CRIM)
 1801, McGill College, suite 800
 Montréal Québec, Canada H3A 2N4

ABSTRACT

Over the past few years, there have been several published reports on the use of MMIE (Maximum Mutual Information Estimation) for training HMM parameters. Many of these reports clearly demonstrate MMIE's capability of substantially reducing the error rate in certain types speech recognition applications. The most convincing demonstrations, however, have usually been carried out on either small vocabulary tasks or on isolated speech. This is in large part a consequence of the fact that, for large vocabularies, MMIE training is much too computationally expensive and approximations must therefore be found in order to bring the task down to a manageable size. This paper looks at such approximations which can be applied in the context of large vocabulary continuous speech recognition and, in particular, proposes a technique based on the use of a compact looped word lattice. Experiments are used to demonstrate the effectiveness of the technique.

Introduction

The use of the so-called Maximum Mutual Information Estimation (MMIE) has often been shown to be effective in substantially improving the recognition performance of HMMs previously trained with the more conventional Maximum Likelihood Estimation (MLE) technique. This is particularly true for small vocabulary continuous speech recognition applications [7].

The idea behind MMIE training is quite simple. Most speech recognition systems use *maximum a posteriori* (MAP) decoding, i.e., given an observed sequence of feature vectors \mathbf{y} and an HMM model \mathbf{m}_w for each possible word sequence \mathbf{w} (typically built from word or sub-word models), decoding is performed by finding the word sequence $\hat{\mathbf{w}}$ such that:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}'}{\operatorname{argmax}} P_{\Theta}(\mathbf{w}'|\mathbf{y}) = \underset{\mathbf{w}'}{\operatorname{argmax}} P_{\Theta}(\mathbf{y}|\mathbf{m}_{\mathbf{w}'})P(\mathbf{w}'), \quad (1)$$

where $P_{\Theta}(\mathbf{y}|\mathbf{m}_w)$ is the probability that the model \mathbf{m}_w produced \mathbf{y} , $P(\mathbf{w}')$ is the *a priori* probability of the word sequence \mathbf{w}' , and Θ is the set of all HMM parameters (transition probabilities and output distribution parameters).

Generally, $P_{\Theta}(\mathbf{y}|\mathbf{m}_w)$ is intended as an estimate of the "true probability" $P(\mathbf{y}|\mathbf{w})$ that the pronunciation of the word sequence \mathbf{w} resulted in \mathbf{y} . This estimate is computed using the model \mathbf{m}_w , whose parameters are, in case of MLE training, estimated by maximizing the following objective function:

$$R(\Theta) = \prod_{r=1}^N P_{\Theta}(\mathbf{y}^r|\mathbf{m}_r), \quad (2)$$

where $\mathbf{m}_r \equiv \mathbf{m}_{w_r}$ is the model corresponding to the word sequence in the r -th training utterance \mathbf{y}^r and $P_{\Theta}(\mathbf{y}^r|\mathbf{m}_r)$ is

the probability that \mathbf{m}_r produced \mathbf{y}^r . Thus, using the models corresponding to the observation sequences, MLE increases the *a posteriori* probability of the training data.

Clearly, however, training should aim at finding an HMM parameter set Θ such that, whenever \mathbf{w} is pronounced, we have $P_{\Theta}(\mathbf{y}|\mathbf{m}_w)P(\mathbf{w}) > P_{\Theta}(\mathbf{y}|\mathbf{m}_{w'})P(\mathbf{w}')$, for any $\mathbf{w}' \neq \mathbf{w}$, as often as possible. It is not intuitively clear how (2) achieves this, especially since the models not corresponding to the training data are not used for parameter estimation. On the other hand, MMIE uses the following objective function:

$$R(\Theta) = \prod_{r=1}^N P_{\Theta}(\mathbf{m}_r|\mathbf{y}^r) = \prod_{r=1}^N \frac{P_{\Theta}(\mathbf{y}^r|\mathbf{m}_r)P(\mathbf{w}_r)}{\sum_{\mathbf{w}'} P_{\Theta}(\mathbf{y}^r|\mathbf{m}_{\mathbf{w}'})P(\mathbf{w}')} \quad (3)$$

That is, MMIE increases the *a posteriori* probability of the model corresponding to the training data, given the data. Since this is also the criterion used in MAP decoding, the relationship between MMIE and error rate is much more intuitive than it is with MLE.

Estimating HMM parameters with MMIE

For several reasons, training HMMs with MMIE is much more complex than it is with MLE. One reason is the non-existence of closed-formed reestimation formulas similar to those available for MLE. One common solution is to resort to some form of gradient descent. We have, however, always preferred to rely upon the reestimation formulas proposed by us in the past [6]. Although not theoretically guaranteed to converge, these formulas have in practice always converged very quickly. This has also been confirmed by other researchers [4]. Either way, however, the value of $\partial \log R(\Theta)/\partial \theta$ must be computed at each iteration for every parameter θ that must be estimated. In order to do so, let us define a model \mathbf{m}_{gen} such that:

$$P_{\Theta}(\mathbf{y}|\mathbf{m}_{gen}) = \sum_{\mathbf{w}'} P_{\Theta}(\mathbf{y}|\mathbf{m}_{\mathbf{w}'})P(\mathbf{m}_{\mathbf{w}'}) \quad (4)$$

That is, (4) replaces the denominator in (3). For every possible path in every possible model \mathbf{m} in the application, there should be a corresponding path, with the same probability, in \mathbf{m}_{gen} . Typically, \mathbf{m}_{gen} will be the model that is searched (implicitly or explicitly) during recognition.

Then, $\partial \log R(\Theta)/\partial \theta$ can be expressed as [7]:

$$\frac{\partial \log R(\Theta)}{\partial \theta} = \frac{1}{\theta} (c_{\theta} - c_{\theta}^{gen}), \quad (5)$$

where c_{θ} represents the standard MLE count for parameter θ

and c_{θ}^{gen} is the same count, but obtained when training is done using m_{gen} .¹ Each iteration of MMIE training can be seen as doing one standard MLE iteration which increments the HMM counts, and one iteration using m_{gen} in place m_r , which decrements these same counts.² The computation time is therefore a direct function of the size of m_{gen} . The problem is of course that, for a typical large vocabulary task, m_{gen} contains several thousand words connected by bigram probabilities and a back-off node. Such a model would have tens of thousands of nodes and would simply be impossible to use.

The remaining of this paper will therefore focus on techniques used to make MMIE training manageable for large vocabulary continuous speech recognition tasks. One important emphasis will be the study of ways to simplify m_{gen} .

The N-Best Approach

One way of simplifying m_{gen} that has been proposed by some authors in the past [2] is to restrict the sum in (4) to the N most likely word sequences w having the highest likelihood. We will refer to this as the N-best approach. It can be seen as learning to discriminate against the most likely, rather than all, competing hypotheses.

Interestingly, this idea has also been proposed as a way of increasing the number of competing hypothesis for some discriminative training techniques [1, 3], which would otherwise typically focus on only one.

In the context of MMIE, each training iteration with the N-best approach would involve the following steps:

- 1) Perform N-best recognition on the training set (e.g., using the tree-trellis algorithm [9] or the word-dependent N-best algorithm [8]).
- 2) Perform a standard MLE training pass using the entire training set to increment the MLE counts. This is called the positive pass.
- 3) Perform another training pass, but this time to decrement the MLE counts using for each utterance a model (called m_r^{Nbest}) having only the N-best hypotheses in parallel. This is called the negative pass. If the correct word sequence is not among the N-best hypothesis, it must be added to m_r^{Nbest} .

Note that step 1 makes the whole procedure quite costly. This cost can however be substantially reduced if we assume that the N-best lists will not change much from one iteration to the next (e.g., if convergence is slow). In this case, step 1 can be done only once every few iterations.

Note also — and this is an important point — that for each training utterance, the correct word sequence will be used in both the positive and negative passes. In the negative pass, the contribution to the counts due to the correct word sequence

¹ Discrete distributions are assumed here, but similar arguments also apply to continuous densities.

² We typically manipulate (5) slightly to improve convergence [7], but this interpretation remains basically valid.

will be weighed by its *a posteriori* probability. Therefore if, for a certain utterance, the probability of any incorrect word sequence is negligible compared to the probability of the correct word sequence (which is what MMIE tries to achieve), then the negative and positive contributions to the counts for this utterance will be almost identical, with the result that this utterance will have an overall negligible contribution to the final counts. It is on that basis that we ignored correctly recognized utterances in the “corrective MMIE training algorithm” [6], introduced some time ago.

The Word Lattice Approach

There are some important problems with the N-best approach to MMIE training. One problem is that nearby hypothesis in N-best lists often only differ by as little as a single word, with the result that many of the branches in m_r^{Nbest} will be nearly identical. This will in turn lead to a large amount of redundant computations.

Another problem is that, in order to fully use the training data, very large N-best lists would be necessary if, for example, it were desired to have competing hypothesis for most words in the training data. This may, however, be computationally difficult, if not impossible.

On the other hand, if short N-best lists are used, they are likely to change significantly from one iteration to the next (due to changes in the models) thereby mandating their recomputation at every iteration, at great computational expenses.

All these problems seem to point towards the use of a word lattice, as has recently become popular in search algorithms (see, e.g., [5]). This is the avenue that we have elected to take. Since, however, there are many different definitions of a word lattice, it seemed appropriate that we proposed one, for our own purposes.

Our lattices are generated using a two-pass (forward-backward) search algorithm in which the first pass is a standard Viterbi beam search which remembers the end word scores of all active words and the backward pass remembers every word transition belonging to a word sequence whose log-likelihood is within a certain Δ of the most likely word sequence. We only remember the two words in the transition (with their pronunciation numbers), and the corresponding log-likelihood. In particular, no time information is kept.

An example lattice is given in Figure 1 for the phrase “FLIGHTS FROM PITTSBURG”. Each line contains information about one lattice entry: an entry number, a lexicon entry (start_end stands for the beginning and end of the lattice and sil stands for the silence model), the lexicon entry number, the pronunciation number, the number of lattice entries this entry connects to, and the list of connections. Each connection is described by a pair of numbers enclosed in brackets; the first is the lattice entry number and the second is the connection log-likelihood (in base 1.001) difference from the best hypothesis (which has a reference likelihood of 0). So for example, this lattice can start with the words ‘FLY’, ‘FLIGHTS’, ‘FLIGHT’, as well as ‘sil’, and the word ‘FLIGHTS’ can be followed by ‘AND’, ‘FROM’, ‘IN’, and ‘TO’.

```

0 start_end -1 0 4 (6, -45828) (5, 0) (4, -24718) (1, -45765)
1 sil 1 0 2 (5, -45765) (0, -33771)
2 PITTSBURGH 1197 0 2 (1, -33771) (0, 0)
3 AND 80 1 1 (2, -36423)
4 FLIGHT 599 0 3 (7, -24718) (8, -49266) (9, -40999)
5 FLIGHTS 600 0 4 (3, -36423) (7, 0) (8, -38870) (9, -39798)
6 FLY 603 0 1 (9, -45828)
7 FROM 631 0 1 (2, 0)
8 IN 729 0 1 (2, -38870)
9 TO 1651 3 1 (2, -39798)

```

Figure 1: Example of a looped lattice for the phrase "FLIGHTS FROM PITTSBURG". For example, this lattice contains (among others) the phrases "FLIGHTS FROM PITTSBURG", "FLIGHT FROM PITTSBURG", "FLIGHTS AND PITTSBURG", "FLIGHTS TO PITTSBURG", "FLY TO PITTSBURG", or even "sil".

It should be remarked that a given pronunciation of a word cannot appear more than once in the lattice. For this reason, and also because these lattices will often loop, we call them *looped lattices*. Note also that the size of the lattice can be controlled not only at the time of its generation but, more importantly, it can be reduced at the time of use by ignoring all word transitions below a certain probability threshold.

MMIE training with the looped lattice approach first requires that a lattice m_r^{lat} be generated for every training utterance r . We assume that the number of different hypotheses in the lattices will be large enough that it will not be necessary to regenerate the lattices after a few training iterations. Then each iteration is done as in steps 2 & 3 of the N-best approach, where m_r^{Nbest} is replaced by m_r^{lat} .

Experiments

The techniques described above were tried in the context of an ATIS (Air Travel Information System) speech recognizer. The task has a vocabulary of around 2000 words and recognition is performed using a bigram language model with a perplexity of around 24. We used a front-end which produces 6 different feature sets: energy and cepstral coefficients, along with their first and second derivatives. Discrete distribution HMMs were used throughout.

One interesting feature about MMIE training is that it is possible for some context-dependent allophone units to be trained, even though these units do not occur in the training data. For example, if standard triphones are used, many triphones will have to be synthesized in order to be able to model all words in the pronunciation lexicon. Some of these words will end up in the word lattice (or the N-best list), thereby training them (albeit with negative counts). Since the number of resulting triphones may be quite large, we have chosen to use allophone units (diphones and tree units) which allow coverage of all possible phonetic contexts with a smaller absolute number of units.

In our experiments, we substantially reduced the computation time by only updating counts using utterances r such that $\log P_{\Theta}(y^r | m_r) - \log P_{\Theta}(y^r | m_{gen})$ is smaller than some threshold (in our case -1). All recognition experiments were

performed with a very tight beam.

Experiments with "1-best"

In this experiment, we used a degenerate version of the N-best approach, in which only the most likely hypothesis was used. It was principally aimed at studying the convergence properties, in a large vocabulary context, of our reestimation formulas. Diphone models were used and MMIE training was done on about one tenth of the ATIS training set. Figure 2 shows convergence results for the objective function (3) with and without codebook exponents [7]. As can be seen, the use of codebook exponents results in a much faster convergence.

Figure 3 shows recognition results on an independent development test set (193 sentences from the Nov92 test set). As can be seen, results with exponents are much better than without, with most of the improvement due to a substantial reduction in the number of insertions.

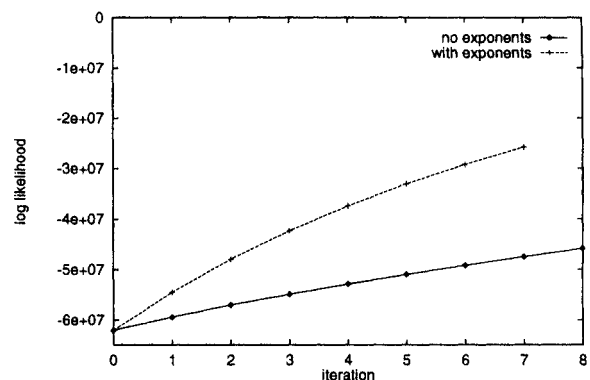


Figure 2: Convergence of the objective function

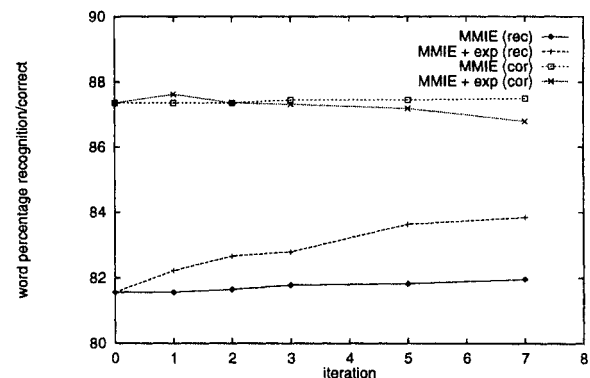


Figure 3: Performance on a small development test set

Experiments with looped lattices

In order to further reduce the size of the looped lattices, training was performed on 3-word sentence fragments rather than whole sentences. This means that the training data had to be pre-segmented accordingly. This segmentation was done automatically and was not verified. Furthermore, cross-word allophonic models were not used during MMIE training. Also, as we often do when performing preliminary

experiments with a new acoustic modeling technique, we left out second derivative parameters (this typically increases the error rate by about 25%).

Two experiments were performed; one which used the language model (LM) probabilities (with the same weighting as in recognition), and one which didn't. Table 1 shows the codebook exponents obtained after 5 MMIE training iterations. Table 2 shows recognition results obtained with various models: baseline (MLE) models, baseline models with codebook exponents from Table 1, and the models obtained after 5 MMIE training iterations (with and without a LM).

Two observations are immediately obvious. First, most of the improvement is due to the codebook exponents. This seems clear from Table 2, where we see that simply using the codebook exponents (trained with a LM) with the MLE models produces most of the improvement obtained with the full MMIE training. Second, using the LM during training has a substantial impact on the final result. In fact, these two observations are linked. Not using the LM during training introduces a mismatch between training and recognition. This results in the algorithm overly reducing the exponents, probably to introduce a kind of word insertion penalty, which offsets the absence of a language model.³

A final interesting observation is that convergence properties are quite different in the two cases (with and without a LM). In particular, convergence with a LM has turned out to be quite slow. We don't yet have a good explanation for that. Moreover, we haven't yet looked at the impact of errors in the automatic segmentation of the training data, but we think it might have a larger impact on MMIE training than on MLE training.

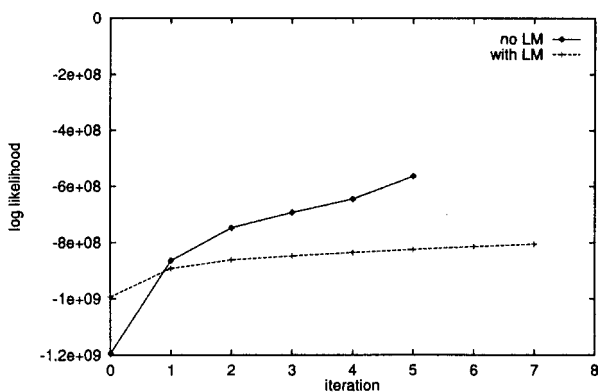


Figure 4: Convergence of the objective function

	C	E	ΔC	ΔE
no LM	0.349920	0.595007	0.769911	0.698807
LM	0.530612	0.411090	1.215430	0.718722

Table 1: Codebook exponents after 5 iterations

³ When codebook exponents decrease, the transition probability contribution to the likelihood computation increases. This increases the cost of rapidly traversing a model and, as a result, tends to decrease the number of insertions.

	W.Err	Corr	Ins	Del	Subs
base	21.8	83.8	214	135	484
base + exp	21.3	81.6	108	218	485
base + exp (LM)	19.5	84.3	147	158	439
mml5	21.1	78.9	105	224	477
mml5 (LM)	19.6	84.5	156	152	438

Table 2: Recognition results on 334 sentences from the Nov92 test set

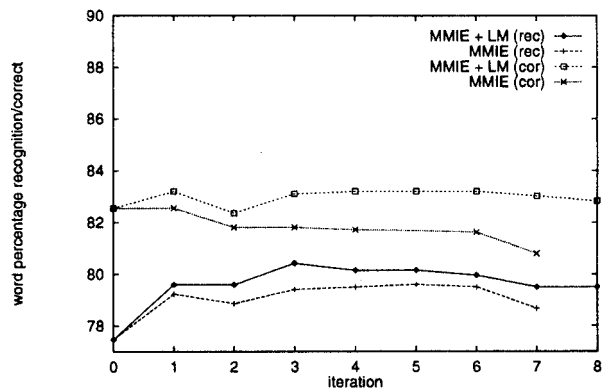


Figure 5: Performance on 101 Nov92 sentences

References

- [1] J.K. Chen and F.K. Soong, "An N-Best Candidates-Based Discriminative Training for Speech Recognition Applications", *IEEE Transactions on Speech and Audio Processing*, vol. 2, no. 1, part II, January 1994
- [2] Y.L. Chow, "Maximum Mutual Information Estimation of HMM Parameters for Continuous Speech Recognition using The N-Best Algorithm", *Proc. ICASSP-90*, paper S13.6, Albuquerque, April 1990
- [3] X. Huang, M. Belin, F. Alleva, and M. Hwang, "Unified Stochastic Engine (USE) for Speech Recognition", *ICASSP-93*, p. II-636, Minneapolis, April 1993
- [4] S. Kapadia, V. Valtchev, and S.J. Young, "MMI Training for Continuous Phoneme Recognition on the TIMIT Database", *ICASSP-93*, p. II-491, Minneapolis, April 1993
- [5] H. Murveit, J. Butzberger, V. Digalakis, and M. Weintraub, "Large-Vocabulary Dictation Using SRI's DECIPHER TM Speech Recognition System: Progressive Search Techniques", *ICASSP-93*, Minneapolis, April 1993
- [6] Y. Normandin and S.D. Morgera, "An Improved MMIE Training Algorithm for Speaker-Independent, Small Vocabulary, Continuous Speech Recognition", *Proc. ICASSP-91*, paper S8.3, pp. 537-540, Toronto, May 1991
- [7] Y. Normandin, R. Cardin, and R. De Mori, "High-Performance Connected Digit Recognition Using Maximum Mutual Information Estimation", *IEEE Transactions on Speech and Audio Processing*, vol. 2, no. 2, April 1994
- [8] R. Schwartz, and S. Austin, "A Comparison of Several Approximate Algorithms For Finding Multiple (N-BEST) Sentence Hypotheses", *ICASSP-91*, Toronto, May 1991
- [9] F.K. Soong and E.-F. Huang, "A Tree-Trellis Based Fast Search for Finding the N Best Sentence Hypotheses in Continuous Speech Recognition", Japan, December 1990