AN IMPROVED CONSENSUS-LIKE METHOD FOR MINIMUM BAYES RISK DECODING AND LATTICE COMBINATION

Haihua Xu[1], Daniel Povey[2], Lidia Mangu[3], Jie Zhu[1]

[1]Department of Electronic Engineering
Shanghai Jiao Tong University,Shanghai,200240,China
[2]Microsoft Research, Redmond, WA, USA
[3] IBM T.J. Watson Research Center, Yorktown Heights, NY
{haihua_xu,zhujie}@sjtu.edu.cn, dpovey@microsoft.com, mangu@us.ibm.com

ABSTRACT

In this paper we describe a method for Minimum Bayes Risk decoding for speech recognition. This is a technique similar to Consensus a.k.a. Confusion Network Decoding, in which we attempt to find the hypothesis that minimizes the Bayes' Risk with respect to the word error rate, based on a lattice of alternative outputs. Our method is an E-M like technique which makes approximations which we believe are less severe than the approximations made in Consensus, and our experimental results show an improvement in WER both for lattice rescoring and lattice-based system combination, versus baselines such as Consensus, Confusion Network Combination and ROVER.

Index Terms— Minimum Bayes Risk (MBR), Consensus, Confusion Network, Speech Recognition, Lattice Rescoring

1. INTRODUCTION

The standard decoding formula used in speech recognition is the Maximum A Posteriori (MAP) rule:

$$W^* = \mathbf{argmax}_W P(W|\mathcal{O}) \quad (1)$$
$$= \mathbf{argmax}_W P(W)p(\mathcal{O}|W), \quad (2)$$

where W is the word-sequence and $\mathcal{O}$ is the observation sequence. This gives the Minimum Bayes Risk estimate with respect to the sentence error. Unfortunately sentence error is not the criterion by which speech recognition systems are usually judged; we typically use a Word Error Rate based on the Levenshtein edit distance [1]. A natural solution to this problem is to use a Minimum Bayes Risk estimate with respect to the word error rate. This is given by:

$$W^* = \mathbf{argmin}_W \sum_{W'} P(W'|\mathcal{O})L(W,W'), \quad (3)$$

where $L(W,W')$ is the Levenshtein edit distance between word sequences W and W' and $P(W'|\mathcal{O})$ is the posterior probability assigned to W' by the model. We would typically limit summations over word sequences to a finite set covered by a lattice. Note that Minimum Bayes Risk decoding is an umbrella term for the various techniques (such as Consensus) that attempt to optimize Equation (3), but it is a slight misnomer because the standard approach of Equation (2) is a Minimum Bayes' Risk estimate with respect to the *sentence* error.

Thanks to Andreas Stolcke and Björn Hoffmeister for help with SRILM tools, and to Geoff Zweig for useful discussions

Sentence	Model Prob
A B C	0.4
A D X	0.3
A D Y	0.3

(a) Modeled probabilities

Reference Sentence	Expected Errors	
	Sent	Word
A B C	0.6✓	1.2
A D C	1.0	1.0 ✓

(b) Expected errors

Fig. 1. Example of MBR estimate differing from MAP estimate

We mention an essential detail at this point: in practice we put a larger weight on the language model log probabilities than on the acoustic model log likelihoods because the latter typically have a very high dynamic range. This weight (typically between 10 to 20) can be applied as a scale on either information source in Equation 2 (MAP decoding), but in Equation (3) it is important to scale down the acoustic likelihoods rather than scale up the likelihoods (e.g. $P(W|\mathcal{O}) \propto P(W)p(\mathcal{O}|W)^\kappa$ with $\kappa = 1/13$ or so). In practice, in our method as in Confusion Networks [2] we generalize this slightly and make κ a separate tunable parameter so we have $P(W)^{\alpha\kappa}p(\mathcal{O}|W)^\kappa$ if α is the normal language model scale (e.g. 12). Note that $P(W)^\alpha$ would also contain the word insertion penalty that is used in some speech recognition systems.

In order to better illustrate the principle of MBR decoding, we give an example (Figure 1) where the MAP and MBR estimates differ. This is similar to the example in [3].

It is quite difficult to do the computation implied in Equation (3). Using lattices and implemented in the obvious way, we would need two nested loops over word sequences W, one for the $\mathbf{argmin}$ and one for the $\sum$, both of which would range over the astronomically large number of word sequences in the lattice, and the inner part where we compute $L(W,W')$ takes time quadratic in the length of the sequences W. Thus some kind of approximation or clever computation is needed.

1.1. Previous work with Minimum Bayes Risk decoding

The first paper to introduce the concept of Minimum Bayes' Error decoding was [3]. In that paper, the $\sum$ was approximated by a long N-best list and it was noted that the $\mathbf{argmin}$ could be approximated by a short N-best list (e.g. 10); $L(W,W')$ was computed using the standard quadratic-time algorithm. That paper was a proof of concept but not practical. Soon afterwards, two different workable approximations were published [2, 4], of which the latter (Consensus a.k.a. Confusion Network Decoding) has become a widely used ap-

proach, and building on it is the approach for system combination known as Confusion Network Combination (CNC) [5], which improves on ROVER [6].

We also note various more recent pieces of relevant work: [7] which exactly optimizes a frame-by-frame word correctness criterion over a lattice, and our own previous work [8] which uses similar approximations as used in Minimum Phone/Word Error (MPE/MWE) training to approximate the summation in (3). This is similar to one of the approaches described in [9]. An interesting piece of related work is [10], applied to MPE/MWE training and using Weighted Finite State Transducer (WFST) based techniques.

2. OVERVIEW OF OUR APPROACH

We use a lattice-based approach that is a hybrid of the forward-backward algorithm on lattices and the basic dynamic-programming edit-distance computation. The algorithm has an estimation phase in which we compute, for each position in the current reference, statistics about the symbol (or no symbol, ϵ) that aligned to it. We accumulate statistics about insertions by including in the reference "dummy" symbols ϵ to which extra words in the lattice can align. We do a kind of discrete E-M where each time we change the reference we can be sure it will decrease the average error. However, there is a slight approximation in the lattice-based recursion such that what we are optimizing is not exactly the same as (3). In a future journal publication we will provide extensive theroetical analysis with proofs.

Note that the time complexity of our algorithm is O of the number of arcs in the lattice times the average number of words of sentences in the lattice. Thus, for a given lattice depth it is quadratic in the length of the utterance. This is asymptotically faster than Consensus, which is cubic in the number of arcs in the lattice.

Before we proceed, in order to establish some notation and motivate the algorithm we write the Levenshtein edit distance [1] computation as a recursion. We have the base cases

$$L(a,a) = 0, a \neq \epsilon \quad (4)$$
$$L(a,b) = 1, a \neq b \quad (5)$$
$$L(\epsilon,a) = 1 \quad (6)$$
$$L(a,\epsilon) = 1 \quad (7)$$
$$L(\epsilon,\epsilon) = 0, \quad (8)$$

where ϵ is a special symbol that means "no symbol" and the last base case is not normally used but is necessary in our approach. The edit distance on strings can then be written as:

$$L(A,B) = \min \left\{ \begin{array}{c} L(a_1,b_1) + L(A_2^{|A|}, B_2^{|B|}) \\ L(a_1,\epsilon) + L(A_2^{|A|}, B) \\ L(\epsilon,b_1) + L(A, B_2^{|B|}) \end{array} \right\} \quad (9)$$

with the base case as $L(\ (),\ ()) = 0$, and where a_1 is the first element of the sequence A and $A_2^{|A|}$ is the rest of the sequence. The simplest way to understand the alignment computation of $L(A,B)$ is to think of an array $L(k,l)$ where $0 \leq k \leq |A|$ is the position in A and $0 \leq l \leq |B|$ is the position in B, and we fill in the array starting from position $(0,0)$; the answer is position $(|A|,|B|)$ in the array.

Before introducing the forward-backward algorithm, we will give an overview of the entire process. It is summarized in Algorithm 1.

Algorithm 1 $R = \text{Rescore}(\mathcal{L})$

1: $R \leftarrow$ MAP (1-best) estimate from lattice $\mathcal{L}$
2: $R \leftarrow \text{normalize}(R)$ // Insert ϵ between each word
3: **while** true **do**
4: Accumulate statistics $\gamma(k,a)$ from $\mathcal{L}$ and R
5: Choose symbol a at each position k in R with highest $\gamma(k,a)$
6: If nothing changed, exit loop.
7: $R \leftarrow \text{normalize}(R)$ // Make sure one ϵ between each word
8: **end while**
9: $R \leftarrow \text{remove-eps}(R)$ // Remove ϵ's between words

3. LATTICE FORWARD-BACKWARD EDIT-DISTANCE COMPUTATION

The core of our algorithm is the forward-backward edit-distance computation, which we break into two separate parts (the forward and backward part) to make the discussion of each part manageable, although they really comprise one algorithm. This entire computation equates to line 4 in Algorithm 1, and its output is the statistics $\gamma(k,a)$ where $1 \leq k \leq |R|$ is a position in the reference and a is a symbol (or ϵ). The algorithm also gives us the expected edit-distance E between the lattice $\mathcal{L}$ and the reference R.

3.1. Inputs

The inputs to the process are a reference R and a lattice $\mathcal{L}$. The reference is a sequence $r_1 \ldots r_{|R|}$. Each element r_i is a symbol $r_i \in \mathcal{S} \cup \{\epsilon\}$, where $\mathcal{S}$ is the set of words and ϵ is a dummy symbol meaning no word is present. The lattice has a number of nodes $N = |\mathcal{L}|$, with nodes $1 \leq n \leq N$. The nodes are assumed to be sorted in topological order, so node 1 is the start node and N the end node (the sorting is necessary). Each node has a set of following arcs $\text{foll}(n)$ and preceding arcs $\text{pred}(n)$. For an arc a, $\text{s}(a)$ is its starting node and $\text{e}(a)$ is its ending node. Each arc has a word label $w(a) \in \mathcal{S} \cup \{\epsilon\}$ (we allow ϵ in order to handle non-scored words such as silence). It also has a "total likelihood" $p(a)$ which is the appropriately weighted product of the acoustic and language model likelihoods, including the scale κ.

3.2. Forward part

The key quantities in our forward algorithm are the standard forward likelihood $\alpha(n)$ for a lattice node n, and the forward average error $\alpha'(n,k)$ for node n and position $0 \leq k \leq |R|$ in the reference. This is a weighted average edit distance up to the current point and is reminiscent of the α' quantity in the MPE computation [11]. We also have the Boolean quantity $b(n,k)$. This does not actually contain any information not present in the α or α' quantities but it is used to speed up the backward computation by remembering certain traceback information. We note that $\alpha(n)$ should be stored as a log quantity in practical implementations. The algorithm below refers to a quantity δ. This is a small positive quantity e.g. $\delta = 10^{-5}$, which is used to break a symmetry and force inserted words to align to ϵ positions in the lattice.

Algorithm 2 can be thought of as a lattice-based version of the recursion of Equation (9), with the first and second recursion options of (9) on line 16 and the third one on line 22. At each point we take a weighted average of incoming paths. There is a slight approximation involved because in effect we are making a single decision how to recurse for whole groups of paths through the lattice rather than

for individual paths. At the end, P is the likelihood of the whole utterance and E is the average approximated edit distance.

Algorithm 2 Forward($\mathcal{L}, R$)

1: $N \leftarrow |\mathcal{L}|$, $K \leftarrow |R|$
2: Initialize arrays $\alpha(1:N)$, $\alpha'(1:N, 0:K)$, $b(1:N, 0:K)$
3: $\forall(n,k)$, $b(n,k) = \text{false}$
4: $\alpha(1) \leftarrow 1$, $\alpha'(1,0) \leftarrow 0$
5: **for** $k \leftarrow 1 \ldots K$ **do**
6: $\quad \alpha'(1,k) \leftarrow \alpha'(1,k-1) + L(\epsilon, r_k)$
7: $\quad b(1,k) \leftarrow \text{true}$
8: **end for**
9: **for** $n \leftarrow 2 \ldots N$ **do**
10: $\quad \alpha(n) \leftarrow \sum_{a \in \text{pred}(n)} \alpha(\text{s}(a))p(a)$
11: $\quad t \leftarrow 0$ // t is only used to control for numerical problems
12: $\quad$ **for** $a \in \text{pred}(n)$ **do**
13: $\quad\quad p \leftarrow \alpha(\text{s}(a))p(a)/\alpha(n)$ // $0 < p \leq 1$
14: $\quad\quad t \leftarrow t + p$
15: $\quad\quad$ **for** $k \leftarrow 0 \ldots K$ **do**
16: $\quad\quad\quad \alpha'(n,k) \mathrel{+}= p \min \left\{ \begin{array}{c} \alpha'(s(a), k-1) + L(w(a), r_k) \\ \alpha'(s(a), k) + L(w(a), \epsilon) + \delta \end{array} \right\}$
$\quad\quad\quad$ // Take $\alpha'(s(a), k-1)$ to be ∞ for $k = 0$
17: $\quad\quad$ **end for**
18: $\quad$ **end for**
19: $\quad \alpha'(n,k) \leftarrow \alpha'(n,k)/t$, $k \in \{0 \ldots K\}$ // Renormalize for numerical reasons; mathematically, t=1.
20: $\quad$ **for** $k \leftarrow 1 \ldots K$ **do**
21: $\quad\quad$ **if** $\alpha'(n,k) > \alpha'(n,k-1) + L(\epsilon, r_k)$ **then**
22: $\quad\quad\quad \alpha'(n,k) = \alpha'(n,k-1) + L(\epsilon, r_k)$
23: $\quad\quad\quad b(n,k) = \text{true}$ // Remember how we recursed
24: $\quad\quad$ **end if**
25: $\quad$ **end for**
26: **end for**
27: $P \leftarrow \alpha(N)$ // P is the total probability of the utterance
28: $E \leftarrow \alpha'(N,K)$ // E is the average approximated error

3.3. Backward part

The next stage of the algorithm (the backward part) accumulates statistics about which symbols aligned to which positions $1 \leq k \leq N$ in the reference R. The statistics are $\gamma(k,a)$ with $a \in \mathcal{S} \cup \{\epsilon\}$ and in practice these would be stored for each position k either as an associative array indexed by a, or as a list of pairs (a,p) which would be sorted on a and the probability summed up for each unique a. The "backward" quantity $\beta(n,k)$ is a backward likelihood comparable to the "forward" $\alpha(n)$, and it should be stored in log form.

4. UPDATE PHASE AND NORMALIZATION

The "update phase" of this process (line 5 of Algorithm 1) is very simple: for each position $1 \leq k \leq |R|$, we take $\hat{r}_k \leftarrow \max_a \gamma(k,a)$, where a will either be a word or ϵ. We can formulate this whole problem as a kind of discrete E-M process; we will not go into details here due to space constraints, but we can prove that the average error E computed in Algorithm 2 will decrease by at least $\sum_k \gamma(k, \hat{r}_k) - \gamma(k, r_k)$.

The normalization process referred to in Algorithm 1 as normalize(R) simply ensures that between every non-ϵ word in R is exactly one ϵ, including at the start and end. This is accomplished by adding and removing ϵ symbols. The process remove-eps(R) simply consists of removing all ϵ symbols.

Algorithm 3 Backward($\mathcal{L}, R, \alpha, \alpha', b, P$)

1: $N \leftarrow |\mathcal{L}|$, $K \leftarrow |R|$
2: Initialize arrays $\beta(1:N, 0:K)$, $\gamma(1:K, \mathcal{S} \cup \{\epsilon\})$.
3: $\forall(n,k)$, $\beta(n,k) \leftarrow 0$
4: $\forall(k,a)$, $\gamma(k,a) \leftarrow 0$ // in practice γ would not be a simple array
5: $\beta(N,K) \leftarrow 1$
6: **for** $n \leftarrow N \ldots 1$ **do**
7: $\quad$ **for** $k \leftarrow K \ldots 0$ **do**
8: $\quad\quad$ **if** $b(n,k)$ **then**
9: $\quad\quad\quad \gamma(k,\epsilon) \leftarrow \gamma(k,\epsilon) + \alpha(n)\beta(n,k)/P$
10: $\quad\quad\quad \beta(n,k-1) \leftarrow \beta(n,k-1) + \beta(n,k)$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: $\quad$ **for** $a \in \text{pred}(n)$ **do**
14: $\quad\quad$ **for** $k \leftarrow 0 \ldots K$ **do**
15: $\quad\quad\quad$ **if** $\neg b(n,k) \wedge \beta(n,k) \neq 0$ **then**
16: $\quad\quad\quad\quad$ **if** $k > 0 \wedge \begin{array}{c} \alpha'(s(a), k-1) + L(w(a), r_k) \leq \\ \alpha'(s(a), k) + L(w(a), \epsilon) + \delta \end{array}$ **then**
17: $\quad\quad\quad\quad\quad \gamma(k, w(a)) \leftarrow \begin{array}{c} \gamma(k, w(a)) + \\ \alpha(s(a))\beta(n,k)p(a)/P \end{array}$
18: $\quad\quad\quad\quad\quad \beta(s(a), k-1) \leftarrow \beta(s(a), k-1) + \beta(n,k)p(a)$
19: $\quad\quad\quad\quad$ **else**
20: $\quad\quad\quad\quad\quad \beta(s(a), k) \leftarrow \beta(s(a), k) + \beta(n,k)p(a)$
21: $\quad\quad\quad\quad$ **end if**
22: $\quad\quad\quad$ **end if**
23: $\quad\quad$ **end for**
24: $\quad$ **end for**
25: **end for**
26: Check that $\beta(1,0) = P$ and $\sum_a \gamma(k,a) = 1$ for $1 \leq k \leq K$.

5. LATTICE COMBINATION

System combination with this scheme is very trivial: we can just assume that we want to find the W^* that minimizes an average of the quantitiy $\sum_{W'} P(W'|\mathcal{O})L(W,W')$ over different systems, i.e.

$$W^* = \mathbf{argmin}_W \sum_{n=1}^{N} \tfrac{1}{N} \sum_{W'} P_n(W'|\mathcal{O})L(W,W'), \qquad (10)$$

where $P_n(W'|\mathcal{O})$ is the posterior in the n'th lattice. We can also consider weighted combinations. The only modification to Algorithm 1 is that we need to average the statistics $\gamma(k,a)$ over the different systems before updating R. In our experiments we picked the best individual system to initialize R.

6. EXPERIMENTAL SETUP

Experiments are performed using the HUB4-98 English broadcast corpus obtained from the LDC, which has about 70 hours of training data after cleaning, and the HUB4-97 corpus which is about 74 hours long. We use two test sets named BN99EN-1 and BN99EN-2, with 1.0 and 1.5 hours respectively.

The data is parameterized as Perceptual Linear Prediction coefficients (PLPs) with energy, plus Δ and $\Delta\Delta$ features yielding a standard 39 dimensional feature vector. Our systems were trained with HTK. We first trained a triphone cross-word MLE system with 3.7k clustered states and 12 Gaussians per state, on HUB4-98 only. This system is referred to as MLE12. In order to have a variety of systems to combine, we used different discriminative training approaches. We trained a system with MPE [11], and two systems with

Boosted MMI [12]; the BMMI01 system was trained in the normal way for 4 iterations and for the BMMI02 system was trained using MPE for one iteration, lattices were regenerated and then we trained for 4 iterations with Boosted MMI. We also trained a larger ML system, on both HUB4-97 and HUB4-98, with 6.3k clustered states and with 16 Gaussians per state, which we refer to as MLE16. Language models are trigram and trained with the HTK tools on the text data provided for the HUB4 task. We used a 60k word dictionary.

Lattices for rescoring were generated with `HDecode` (note that this is the HTKV3.4 version of `HDecode` which has improved lattice generation). We tested with language model scale 12.0 and log insertion probability -10.0 (i.e. insertion penalty 10). We did experiments with two scales κ: 0.065 which had been optimized for MPE training, and 0.0833 which was the inverse of the language model scale. Confusion network experiments worked better with 0.0833 and our approach with 0.065 so we chose those settings. After tuning the pruning beam in the confusion network computation we chose 25 (applied before κ). We worked with two Consensus/CNC implementations: our own, and the SRILM toolkit [13], creating the CNs with `lattice-tool -write-mesh` and combinng them with `nbest-lattice -use-mesh -lattice-files ...`. We had the best results with our own implementation for single system decoding, and SRILM for CNC; we report these results. For results with our method reported here, we iterated our algorithm for three iterations; typically if allowed to run till completion the algorithm takes two to four iterations.

7. RESULTS

System	Decoding	BN99EN-1	BN99EN-2	Overall
MLE12	MAP baseline	31.51	29.61	29.39
	Consensus	30.24	28.34	29.11
	Proposed method	29.99	28.18	28.92
MLE16	MAP baseline	28.42	26.96	27.56
	Consensus	28.15	26.62	27.25
	Proposed method	28.07	26.58	27.19
MPE	MAP baseline	27.92	25.74	26.64
	Consensus	27.46	25.42	26.26
	Proposed method	27.35	25.25	26.11
BMMI01	MAP baseline	28.60	26.17	27.17
	Consensus	28.34	25.93	26.92
	Proposed method	28.14	25.69	26.70
BMMI02	MAP baseline	27.66	25.70	26.50
	Consensus	27.31	25.24	26.09
	Proposed method	27.18	25.14	25.98

Table 1. Single-system lattice rescoring: %WER

Table 1 shows traditional (MAP) decoding versus Consensus and our proposed method, for the four models and two different test sets. In each case Consensus gives an improvement over the baseline, and our method gives a further improvement. The average relative WER reduction is 1.2% from Consensus and 1.7% from our proposed method.

For system combination experiments we experimented with two different scenarios: three-system combination and four-system combination, as seen in Tables 2 and 3. We applied ROVER to the various individual results of Table 1 and also used CNC and our approach of Section 5. In all cases our system combination method is slightly better than Consensus/CNC.

Combination Method	BN99EN-1	BN99EN-2	Overall
Pick best system	27.66	25.70	26.50
MAP+ROVER	27.63	25.30	26.26
CN+ROVER	27.36	25.11	26.03
Proposed+ROVER	27.04	24.94	25.80
CNC	27.09	25.03	25.88
Proposed	26.91	24.84	25.69

Table 2. 3-system combination: MPE, BMMI01, BMMI02: %WER

Combination Method	BN99EN-1	BN99EN-2	Overall
Pick best system	27.66	25.70	26.50
MAP+ROVER	27.36	25.40	26.20
CN+ROVER	27.31	25.23	26.09
Proposed+ROVER	27.13	25.00	25.88
CNC	26.49	24.54	25.34
Proposed	26.25	24.46	25.20

Table 3. 4-system combination: MPE, BMMI01, BMMI02, MLE16

8. CONCLUSIONS

We have introduced a Minimum Bayes Risk decoding technique for lattice rescoring and system combination. It has similar functionality to Consensus, but is simpler to implement. Based on the experiments reported here, it seems to give slightly better results than Consensus/CNC in both lattice rescoring and system combination scenarios. We believe it is a good replacement for Consensus/CNC because it is simpler and (as we will describe in future) has a clearer theoretical basis.

9. REFERENCES

[1] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, 1966.

[2] L. Mangu, E. Brill and A. Stolcke, "Finding consensus in speech recognition: word error minimization and other applications of confusion networks," *Computer Speech and Language*, vol. 14, no. 4, 2000.

[3] Y. König A. Stolcke and M. Weintraub, "Explicit word error minimization in N-best list rescoring," in *5th European Conf. on Speech Comm. and Technology*, 1997, pp. 163–166.

[4] V. Goel and W. Byrne, "Minimum Bayes-risk automatic speech recognition," *Computer Speech and Language*, vol. 14, no. 2, 2000.

[5] G. Evermann and P.C. Woodland, "Posterior probability decoding, confidence estimation and system combination," in *Speech Transcription Workshop*, 2000.

[6] J. G. Fiscus, "A post-processing system to yield reduced word error rates: Recognizer Output Voting Error Reduction (ROVER)," in *ASRU*, 1997.

[7] F. Wessel, R. Schlüter, H. Ney, "Explicit word error minimization using word hypothesis posterior probabilities," in *ICASSP*, 2001.

[8] Xu H., Povey D., Zhu J. and Wu G., "Minimum Hypothesis Phone Error as a Decoding Method for Speech Recognition," in *Interspeech*, 2009.

[9] Björn Hoffmeister, Ralf Schlüter, and Hermann Ney, "Bayes Risk Approximations Using Time Overlap with an Application to System Combination," in *Interspeech*, 2009.

[10] Heigold G., Macherey W., Schlüter R., Ney H., "Minimum Exact Word Error Training," in *ASRU*, 2005.

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

[12] 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.

[13] A. Stolcke, "SRILM - An Extensible Language Modeling Toolkit," in *ICSLP*, 2002.