Lattice Decoding and Rescoring with Long-Span Neural Network Language Models

Martin Sundermeyer[1], *Zoltán Tüske*[1], *Ralf Schlüter*[1], *Hermann Ney*[1,2]

[1]Human Language Technology and Pattern Recognition,
Computer Science Department, RWTH Aachen University, Aachen, Germany
[2]Spoken Language Processing Group, LIMSI CNRS, Paris, France
{sundermeyer,tueske,schlueter,ney}@cs.rwth-aachen.de

Abstract

With long-span neural network language models, considerable improvements have been obtained in speech recognition. However, it is difficult to apply these models if the underlying search space is large.

In this paper, we combine previous work on lattice decoding with long short-term memory (LSTM) neural network language models. By adding refined pruning techniques, we are able to reduce the search effort by a factor of three.

Furthermore, we introduce two novel approximations for full lattice rescoring, which opens the potential of lattice-based speech recognition techniques. Compared to 1000-best lists, we find that we can increase the word error rate improvements obtained with LSTMs from 8.2 % to 10.7 % relative over a state-of-the-art baseline, while the resulting lattices are even considerably smaller. In addition, we investigate the use of LSTMs for Babel Assamese keyword search, obtaining significant improvements of 2.5 % relative.

Index Terms: speech recognition, language modeling, recurrent neural networks, long short-term memory, word lattices

1. Introduction

In today's speech recognition systems, the language model (LM) estimates the probability $p(w_1^N)$ of a word sequence w_1^N, where a conditional dependence on the previous $(n-1)$ words is assumed. This results in the factorization

$$p(w_1^N) = \prod_{i=1}^{N} p(w_i|w_{i-n+1}^{i-1}).$$

Most commonly, the quantity $p(w_i|w_{i-n+1}^{i-1})$ is estimated using a count-based model [1], and it is found that no improvements can be obtained by going beyond $n = 4$ or $n = 5$.

More recently, language models based on neural networks ([2, 3, 4, 5]) have proven especially effective for speech recognition. For this kind of model, much longer context dependences are beneficial. E. g., in [6], a feedforward neural network LM with $n = 8$ was used, and in case of recurrent neural networks ([4, 6]), there is not even an explicit upper bound on the context length that is used for the estimation of the word posterior probabilities.

For efficiency reasons, neural network LMs are mainly applied in a second decoding pass. This means that a set of candidates for the correct word sequence is obtained using a count-based LM. In a second pass, the neural network LM is evaluated on the set of candidates, and the best hypothesis according to the new LM is selected.

In this paper, we stick to the following terminology: By *decoding*, we denote the process of obtaining the overall best hypothesis from a set of candidate hypotheses. We refer to the term *rescoring* in the case where we obtain updated probability estimates for each hypothesis from the candidate set.

The mismatch in context length of the count-based LM for the first pass and the neural network LM for the second pass makes it difficult to apply the neural network in practice. More precisely, if the set of hypotheses is encoded as an n-best list, both decoding and rescoring are straightforward and can be carried out without introducing approximations ([7, 8]). However, only a small fraction of the overall search space can be covered. Even for large n-best lists, most hypotheses will differ in a few word positions only, such that there is little variation in the list.

If the set of hypotheses is encoded as a word lattice, the underlying search space can be represented accurately. On the other hand, for decoding, only an approximative approach is possible, because evaluating the neural network LM for all the unique (with respect to the neural network LM) paths in the lattice is computationally too costly. In [9], a hill climbing solution was used for lattice decoding. In [10], an efficient push forward algorithm was presented to find the best path in a word lattice in a machine translation setting.

To the best of our knowledge, no prior work has investigated the problem of rescoring lattices with a long-span neural network LM. This is important, because many speech recognition techniques rely on lattices, e. g., word confidence estimation, confusion network based decoding, and keyword search. In all these cases, using an n-best list instead of a lattice would result in a severe degradation in performance. In particular, with existing approaches, the improvements obtained with recurrent neural networks (which were shown to outperform feedforward models in [6, 11, 12]) cannot be transferred to any of the above applications.

This paper introduces the following novelties: First, we apply the push forward algorithm from [10] to a speech recognition setting, and we combine it with long-span recurrent long short-term memory (LSTM) neural network LMs ([5, 13]). Second, we investigate refined pruning techniques. Finally, we propose two novel approximation strategies for full lattice rescoring with recurrent neural networks, and evaluate their effectiveness for French speech recognition and Babel Assamese keyword search.

2. LSTM Neural Network LMs

There exist multiple variants of neural network LMs. In this work, we make use of an architecture that is depicted in Fig. 1, consisting of four layers (including input and output layer). This results in the following equations for the forward pass of the neural network:

$$\begin{aligned}
y_{i-1} &= A_1 x_{i-1} \\
z_{i-1} &= \xi(y_{i-1}; A_2, y_1^{i-2}) \\
p(c(w_i)|w_1^{i-1}) &= \varphi_{c(w_i)}(A_3 z_{i-1}) \\
p(w_i|c(w_i), w_1^{i-1}) &= \varphi_{w_i}(A_{c(w_i)} z_{i-1}) \\
p(w_i|w_1^{i-1}) &= p(w_i|c(w_i), w_1^{i-1}) \cdot p(c(w_i)|w_1^{i-1})
\end{aligned}$$

Here, by x_{i-1} we denote the one-hot encoded vector representation of the most-recent history word w_{i-1}, and y_{i-1} and z_{i-1} are the outgoing activation values of the projection layer and the hidden layer, respectively. The matrices $A_1, A_2, A_3, A_{c(w_i)}$ contain the weights of the corresponding neural network layers. By $\xi(\cdot\,; A_2, y_1^{i-2})$ we denote the LSTM formalism that we plug in at the third layer. As the LSTM layer is recurrent, we explicitly include the dependence on the previous layer activations. The equations corresponding to ξ can be found e. g. in [14]. Finally, φ is the widely-used softmax function to obtain normalized probabilities, and c denotes a word class mapping from any vocabulary word to its (unique) word class that we use to speed up the computations ([15, 16]). Word classes can be trained based on a perplexity criterion ([17, 18]).

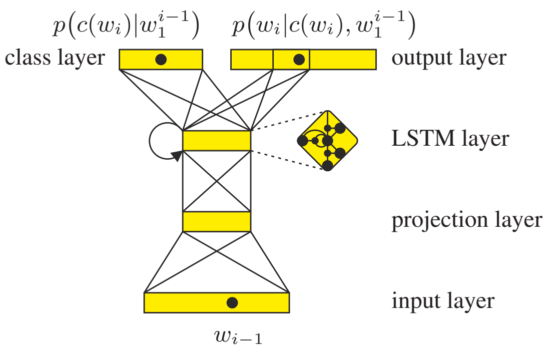

Figure 1: Architecture of the recurrent LSTM neural network LM used throughout this work.

3. Lattice Decoding

3.1. Push Forward Algorithm

As our work builds on the push forward algorithm from [10], we shortly sketch the idea of this approach: In a first step, the nodes of a lattice are sorted in topological order. Then the neural network language model is evaluated, starting from the first node, computing the probability for each word label on the outgoing arcs of the current node, then iterating over the successor nodes.

As the order of the neural network LM is higher than that of the LM which was used to create the lattice, paths in the lattice may be recombined even though they would need to be kept separate in terms of the neural network LM. Therefore, a set of partial hypotheses is associated with each lattice node, where a hypothesis keeps track of the full LM context. To limit the number of hypotheses during decoding, out of those hypotheses where the $(n-1)$ predecessor words of the LM contexts match, only the best hypothesis is retained at a lattice node (n-gram recombination). In addition, only the m-best hypotheses are retained at each node (cardinality pruning).

3.2. Refined Pruning Strategies

We found that we can easily transfer pruning techniques from standard speech recognition decoders to the case of the push forward algorithm.

This relates to the fact that in a speech recognition lattice, multiple nodes share a common time stamp, unlike in the case of machine translation. Sorting the lattice nodes by time and traversing them in this order leads to a (topological) time-synchronous expansion of the search graph. A time-synchronous processing was found to be very efficient for beam pruning in case of first pass speech decoding ([19]).

Consequently, for a given time stamp in the lattice, we compute the best score of all related hypotheses. Afterwards, we discard those hypotheses whose score exceeds the current best score augmented by a fixed threshold. (By *score* we denote the negative logarithm of the corresponding probability.)

In first pass decoding, it is also common to use some kind of look ahead technique regarding scores that will be computed in the future. For example, in first pass decoders relying on a prefix tree representation of the lexicon, the best reachable LM score is added to the scores computed for the current time frame ([19]). By contrast, for transducer-based decoders, the sum over all reachable scores is taken into account instead, and it is even found that a look ahead based on the single best path can decrease performance ([20]).

From this perspective, the single-best look ahead and the sum look ahead seem promising in case of lattice decoding, and we investigate both in the experimental section. In either case, we make use of the scores from the acoustic model as well as from the count-based LM, which both are already available at lattice decoding time.

It should be added that beam pruning is a necessary requirement for look ahead techniques. When using cardinality pruning instead, only the scores of hypotheses attached to a single node are compared with each other. As a result, look ahead does not have any effect because it is the same for all hypotheses.

4. Lattice Rescoring

During lattice decoding, we keep track of all partial paths that have been considered as potential candidates for the overall best path by using a so-called *traceback* data structure as is used in conventional first pass decoding ([19]). In this way, we build up a tree structure that represents all hypotheses considered during decoding. An example is shown in Fig. 2, where arcs from the lattice are shown in black, and arcs from the corresponding traceback tree are drawn in red.

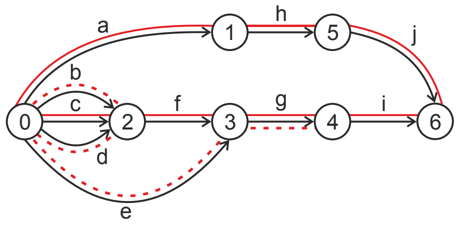

Figure 2: Example lattice and hypothetical traceback tree (indicated in red). Dashed lines correspond to pruned paths.

Our goal is to make use of the probabilities computed during decoding to obtain a lattice that incorporates the neural network LM probabilities.

4.1. Replacement Approximation

For the first approximation, we keep the lattice structure fixed, i. e., we only exchange the LM probability estimates themselves. Furthermore, we choose the approximation in such a way that in case where the lattice would be fully expanded (i. e., correspond to an n-best list, be it compressed as a prefix tree or not) it would result in an exact rescoring.

To this end, we sort all traceback hypotheses decreasingly by time, and increasingly by their score. Afterwards, starting from the final node, we follow the best path back to the start node, assigning the neural network probabilities for the individual words as new LM scores to the visited arcs. In the example of Fig. 2, this would be, say, the path i–g–f–c. We continue to follow back the other traceback paths from the final node, leaving unchanged the scores of lattice arcs that have already been rescored before. Then, we continue in the same manner with the hypotheses that, due to pruning, ended at an earlier node (paths g–e, b, and d in the example lattice). We enforce that at each lattice node at least one hypothesis survives pruning. Like this, we are guaranteed that we can fully rescore the original lattice.

4.2. Traceback Lattice Approximation

A disadvantage of the replacement approximation is that only a part of the neural network probabilities is actually kept for building the rescored lattice. Instead, we can also allow the final lattice to have a structure different from the original one. We choose our approximation in such a way that the overall best path in the rescored lattice must be the same as the one that is found by lattice decoding.

To achieve this goal, we directly convert the traceback tree back into a lattice. However, there is a huge difference in the number of paths in the lattice and its decoding traceback: A typical lattice in our setup contains 10^{35} different paths from the start node to the final node. By contrast, the traceback tree only contains about 100 of such paths. All missing paths in the traceback have been pruned, and we recover them by recombining pruned paths with surviving (but not necessarily complete) paths in the traceback tree. To do so, for each pruned traceback path ending at a lattice node, we determine the worst traceback path surviving this node which is still better than the pruned path. (Due to recombination pruning (see Section 3.1), a surviving path is not necessarily better than a pruned path.) Then we recombine the pruned and the surviving path. In the example in Fig. 2, this means that the distinct paths e–g and c–f–g will meet after the former path was pruned, and they continue as one path starting from node 4.

The resulting lattice has the exact same number of paths as the original lattice. In contrast to the situation of the push forward algorithm, it is not expanded to a unique context up to a certain length.

5. Experimental Results

5.1. French Speech Recognition

The LM training data are summarized in Table 1. The acoustic model for the French experiments was trained on 350 hours of broadcast conversational and broadcast news data. In addition, we incorporated multilingual multi-layer perceptron (MLP) features ([22]) in a tandem ([23]) approach. The features were trained on a total of 840 hours of English, French, German, and Polish speech data. The system made use of discriminative training with the minimum phone error criterion ([21]). Including rescoring 100-best lists with an LSTM with 300 hidden nodes, this system obtained the best word error rate in the final evaluation of the Quaero project[1].

Corpus		Running Words	Vocabulary
Quaero French	Train	100 M	188 K
	Dev	35 K	
	Test	41 K	
Babel Assamese	Train	406 K	22 K
	Dev	66 K	

Table 1: Training data for the French and Assamese languages.

It seems interesting to analyze the explicit context dependence of a recurrent LSTM neural network LM for lattice decoding. Fig. 3 depicts the word error rate dependence on the recombination order at a constant, loose beam pruning threshold. We observe that at an order of 9 (i. e., a 10-gram), the improvement in log-probability saturates. The behaviour of the word error rate is a bit more noisy, but essentially reflects the tendency of the log-probability curve. For the subsequent experiments, we thus stick to a 10-gram recombination pruning.

From the results shown in Figure 4 we conclude that beam pruning for lattice decoding can significantly reduce the search space. When combining beam search with look ahead, we find that, for obtaining the best word error rate, we can reduce the search space as well as decoding time by more than a factor of three compared to cardinality pruning. In this case, the real time factor is 0.5 on one Intel Westmere CPU core.

Table 2 summarizes the results for decoding and rescoring, where the pruning parameters are kept fixed at loose values. In the following, we discuss the results from the table row by row. Regarding the Kneser-Ney baseline, we see that we obtain good improvements of 0.3 % absolute on the dev data and 0.5 % absolute on the test data by using confusion network (CN) decoding instead of Viterbi (V) decoding. We can significantly reduce this error rate by rescoring 100-best lists with an LSTM, but no additional gain is observed with CN decoding. Even when

[1] http://www.quaero.org

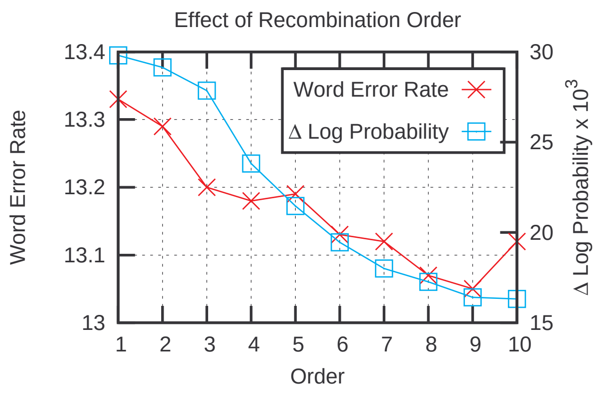

Figure 3: Word error rates for different recombination orders, and change in log probability of the best path.

Type	Development Data							Test Data						
	Dens.	Decode	Repl. Appr.		Traceback Appr.			Dens.	Decode	Repl. Appr.		Traceback Appr.		
			V	CN	Dens.	V	CN			V	CN	Dens.	V	CN
KN4	*124*	14.4		14.1				*166*	16.4		15.9			
100-best	*128*	13.4		13.4				*103*	14.8		14.8			
1000-best	*1289*	13.2		13.1				*1100*	14.7		14.6			
Lattice-1	*124*	13.1	13.2	12.9	*592*	13.1	12.6	*166*	14.6	14.8	14.4	*828*	14.6	14.2
Lat.-4 dep.	*392*	12.8	12.9	12.6	*880*	12.8	**12.5**	*523*	14.4	14.7	14.3	*1209*	14.4	**14.1**

Table 2: Word error rate results for Quaero. If no error rate is given, it is identical to the corresponding Decoding/CN result of the same row. The Kneser-Ney 4-gram (KN4) was trained on 1.6 B running words. Lattice density (Dens.) is measured in arcs per reference word. Lattices after rescoring with the replacement approximation do not increase in size, as opposed to the traceback approximation.

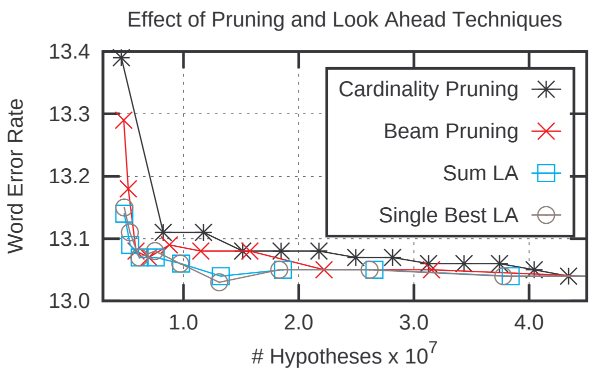

Figure 4: Tradeoff between the number of neural network probability calculations and word error rate performance for various pruning and look ahead (LA) techniques.

switching to large n-best lists of size 1000, the improvement by CN decoding is negligible.

For lattices, we distinguish between direct Viterbi decoding with an LSTM, and two step decoding with an intermediate lattice rescoring, either with the replacement or with the traceback approximation. We see that the Viterbi word error rate of the replacement approximation falls a bit behind the error rate of a direct Viterbi decoding, but with CN decoding, in spite of the approximation we can even improve over direct decoding.

For traceback rescoring, by construction we obtain the same Viterbi word error rate as for direct decoding, and with CN decoding, we improve by 0.5 % absolute on the dev data and 0.4 % absolute on the test data, while the resulting lattices are still considerably smaller than the corresponding 1000-best lists (99 % confidence intervals as in [24]: (0.2 %, 0.7 %) and (0.1 %, 0.6 %), respectively).

The RWTH speech decoder [25] generates lattices which are not expanded by default. Expanding the lattices to a unique 4-gram context, and re-using the LSTM history of the last speech paragraph for the current one in an inter-dependent fashion results in the lowest overall word error rate, an 11.3% relative improvement over the Kneser-Ney model with CN decoding (from 14.1 % to 12.5 % on the development data, and from 15.9 % to 14.1 % on the test data).

We did not optimize the pruning parameters such that the size of the traceback lattices is minimized. For loose pruning parameters without look ahead, the traceback lattices were always smaller than 1000-best lists (except for traceback lattices created from the lattices expanded to a unique 4-gram context.)

5.2. Babel Assamese Keyword Search

We also investigated the potential improvements in keyword search by lattice rescoring with LSTM LMs. It is state-of-the-art to use a bigram count-based LM ([26]) for keyword search. Furthermore, best results are obtained for extremely huge lattices. Therefore we only investigate the replacement approximation here.

We took our current best system for Babel Assamese keyword search system trained on the full language pack according to the 'BaseLR' conditions as a baseline for this experiment. The lattices had 7900 arcs per second. Details can be found in [27]. For the baseline system, we obtained a maximum term-weighted value (MTWV) score of 0.4936 (cf. Tab. 3). By interpolating with four bigram MLP neural network language models (which would reflect the order of the original count-based LM), we improve this MTWV score to 0.5001. If we exchange the MLP by an LSTM instead, we can finally increase this value to 0.5060, improving the baseline score by 2.5 % relative.

LM	MTWV Score
KN Bigram	0.4936
+4 x Bigram MLP	0.5001
+4 x LSTM	0.5060

Table 3: Keyword search results for Babel Assamese.

6. Conclusion

In this work, we investigated a lattice decoding algorithm on two speech recognition setups. We refined previous pruning techniques, leading to a three times reduction in search effort. With the introduction of two novel approximation techniques for lattice rescoring, we were able to reduce the word error rate of a state-of-the-art French speech recognition system with an LSTM by 10.7 % relative, instead of 8.2 % when using 1000-best lists, and our rescored lattices are still considerably smaller. Finally, our rescoring approach also helps improving Assamese keyword search.

7. Acknowledgements

We thank David Nolden for helpful discussions. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreements nos. 287658 (EU-Bridge) and 287755 (transLectures). H. Ney was partially supported by a senior chair award from DIGITEO, a French research cluster in Ile-de-France. This effort uses the IARPA Babel Program Assamese language collection release IARPA-babel102b-v0.5a. Experiments were performed with computing resources granted by JARA-HPC from RWTH Aachen University under project 'jara0085'.

8. References

[1] Kneser, R., and Ney, H., "Improved Backing-Off For M-Gram Language Modeling", Proc. of ICASSP 1995, pp. 181–184

[2] Bengio, Y., and Ducharme, R., "A neural probabilistic language model", Proc. of Advances in Neural Information Processing Systems (2001), vol. 13., pp. 932–938

[3] Schwenk, H., "Continuous Space Language Models", Computer Speech and Language 21 (2007), pp. 492–518

[4] Mikolov, T., Karafiát, M., Burget, L., Černocký, J., and Khudanpur, S., "Recurrent neural network based language model", Proc. of Interspeech 2010, pp. 1045–1048

[5] Sundermeyer, M., Schlüter, R., and Ney, H., "LSTM Neural Networks for Language Modeling", Proc. of Interspeech 2012

[6] Sundermeyer, M., Oparin, I., Gauvain, J.-L., Freiberg, B., Schlüter, R., and Ney, H., "Comparison of Feedforward and Recurrent Neural Network Language Models", Proc. of ICASSP 2013, pp. 8430–8434

[7] Kombrink, S., Mikolov, T., Karafiát, and Burget, L., "Recurrent neural network based language modeling in meeting recognition", Proc. of Interspeech 2011, pp. 2877–2880

[8] Si, Y., Zhang, Q., Li, T., Pan, J., and Yan, Y., "Prefix Tree based N-best list Re-scoring for Recurrent Neural Network Language Model used in Speech Recognition System", Proc. of Interspeech 2013, pp. 3419–3423

[9] Deoras, A., Mikolov, T., and Church, K., "A Fast Re-scoring Strategy to Capture Long-Distance Dependencies" Proc. of EMLNP 2011, pp. 1116-1127

[10] Auli, M., Galley, M., Quirk, C., and Zweig, G., "Joint Language and Translation Modeling with Recurrent Neural Networks", Proc. of EMNLP 2013, pp. 1044–1054

[11] Mikolov, T., Kombrink, S., Burget, L., Černocký, J., and Khudanpur, S., "Extensions of Recurrent Neural Network Language Model", Proc. of ICASSP 2011, pp. 5528–5531

[12] Arısoy, E., Sainath, T. N., Kingsbury, B., and Ramabhadran, B., "Deep Neural Network Language Models", Proc. of NAACL-HLT 2012 Workshop, pp. 20–28

[13] Hochreiter, S., and Schmidhuber, J., "Long Short-Term Memory", Neural Computation 9 (8), 1997, pp. 1735–1780

[14] Graves, A., Mohamed, G., and Hinton, G., "Speech Recognition with Deep Recurrent Neural Networks", Proc. of ICASSP 2013, pp. 6645–6649

[15] Goodman, J., "Classes for fast maximum entropy training", Proc. of the ICASSP 2001, pp. 561–564

[16] Morin, F., and Bengio, Y., "Hierarchical Probabilistic Neural Network Language Model", Proc. of the 10th Int. Workshop on Artificial Intelligence and Statistics 2005, pp. 246-252

[17] Kneser, R., and Ney, H., "Forming Word Classes by Statistical Clustering for Statistical Language Modelling", Proc. of QUALICO 1991, pp. 221–226

[18] Brown, P. F., deSouza, P. V., Mercer, R. L., Della Pietra V. J., and Lai, J. C., "Class-Based n-gram Models of Natural Language", Computational Linguistics 18 (4), pp. 467–479

[19] Ney, H., and Ortmanns, S., "Progress in Dynamic Programming Search for LVCSR", Proc. of the IEEE, Vol. 88, No. 8, 2000, pp. 1224–1240

[20] Mohri, M., Pereira, F., and Riley, M., "Speech Recognition with Weighted Finite-State Transducers", in Springer Handbook on Speech Processing and Speech Communication, Springer, 2008, pp. 559–584

[21] Povey, D., and Woodland, P. C., "Minimum Phone Error and I-smoothing for improved discriminative training", Proc. of ICASSP 2002, pp. 105–108

[22] Tüske, Z., Schlüter, R., and Ney, H., "Multilingual Hierarchical MRASTA Features for ASR", Proc. of Interspeech 2013, pp. 2222–2226

[23] Hermansky, H., Ellis, D. P., and Sharma, S., "Tandem connectionist feature extraction for conventional HMM systems", in Proc. of ICASSP 2000, pp. 1635–1638

[24] Bisani, M., and Ney, H., "Bootstrap Estimates for Confidence Intervals in ASR Performance Evaluation", Proc. of ICASSP 2004, pp. 409–412

[25] Rybach, D., Gollan, C., Heigold, G., Hoffmeister, B., Lööf, J., Schlüter, R., and Ney, H., "The RWTH Aachen University Open Source Speech Recognition System", Proc. of Interspeech 2009, pp. 2111–2114

[26] Knill, K. M., Gales, M. J. F, Rath, S. P., Woodland, P. C., Zhang, C., Zhang, S.-X., "Investigation of multilingual deep neural networks for spoken term detection", Proc. of ASRU 2013, pp. 138–143

[27] Tüske, Z., Nolden, D., Schlüter, R., and Ney, H., "Multilingual MRASTA Features for Low-resource Keyword Search and Speech Recognition Systems", Proc. of ICASSP 2014, pp. 7904–7908