FASTER SEQUENCE TRAINING

Albert Zeyer, Ilia Kulikov, Ralf Schlüter, Hermann Ney

Human Language Technology and Pattern Recognition, Computer Science Department,
RWTH Aachen University, 52062 Aachen, Germany

{zeyer, kulikov, schlueter, ney}@cs.rwth-aachen.de

ABSTRACT

It has been shown that sequence-discriminative training can improve the performance for large vocabulary continuous speech recognition. Our main contribution is a novel method for reducing the computation time of any sort of sequence training while only slightly decreasing the overall performance. The method allows to parallelize the forward propagation through the network, the loss and loss gradient calculation which will provide a frame-wise error signal, and an independent forward and back propagation using that error signal. That last step can be calculated in a frame-wise manner and thus allows to use frame chunking to further improve the runtime. The loss calculation can itself be parallelized over many sequences. In addition to several experiments which outline the runtime gains, we also provide a convergence proof sketch. We extend on the research of sequence training of bidirectional long-short term memory ((B)LSTM) networks and provide an overview and comparison over different criteria. We have published all the code as part of our RETURNN and RASR framework including our training setup configurations.

Index Terms— sequence training, LSTM, computation time, parallelization

1. INTRODUCTION

Conventionally, for simplicity and speed-performance, we train neural networks based on the frame-wise cross-entropy criterion, at least as a first pass. Nevertheless, the evaluation measure for ASR is word error rate (WER). Therefore, an ideal training criterion would aim for minimum WER on the training data.

A sequence-discriminative training criteria such as Minimum Phone Error (MPE) or Maximum Mutual Information (MMI) aim to consider hidden Markov model (HMM) constraints, lexicon and language model such that training correlates more with the WER optimization than frame-wise criteria. It is often performed as the final step after frame-wise training because it requires lattices obtained from the baseline model. From the literature, we can expect about 5-20% relative improvement in WER when applying sequence training on a frame-wise trained model [1, 2].

We address the calculation runtime of sequence training and present a novel method which can be applied to any sequence-level criterion which enables faster calculation by parallelizing parts of the calculation needed during sequence training. We provide an analysis of the method, followed by experimental results in order to demonstrate the speed gains and the WER performance gains obtained.

Furthermore we provide comparisons over popular sequence-discriminative criteria and study other aspects such as the language model influence.

2. RELATED WORK

Sequence-discriminative training was already applied to GMM-HMM-based systems [3, 4]. Later, it was shown that it can also be applied on hybrid HMM-NN systems [5, 6, 7, 8, 1]. Other related work involving sequence training with (B)LSTM-HMM can be found in [9, 10, 11]. In this work we extend the results with deep BLSTM-HMM sequence training.

Recently, it was also shown how to train a model with MMI criterion without lattice, thus it enables to apply sequence training from scratch, i.e. without a frame-wise baseline [12].

Another sequence criterion which is often used and which can also be used directly without a frame-wise pretraining step is the connectionist temporal classification (CTC) criterion [13], although CTC-training is usually still followed by MBR at the end [14, 15, 16]. Our parallel training method can also be applied to CTC and lattice-free MMI.

To the best of our knowledge, there is not much research done about speeding up the sequence training methods. The authors of [17, 9] apply asynchronous stochastic gradient descent which is another form of parallelization which can be applied in combination with our method. Our method even works with a single copy of the model parameters.

3. SEQUENCE TRAINING

In this work we use a hybrid-HMM with deep BLSTM network as the acoustic model. We prefer MPE based criteria like Minimum Phone Error (MPE) and state-level Minimum Bayes Risk (sMBR) which showed to outperform MMI criterion [1, 8], cf. Table 3. Further, we use the following notation: r is the spoken utterance, X_r is the acoustic observation for utterance r, W_r is the word sequence for utterance r, and S_r is the sequence of HMM states corresponding to W_r.

The MMI criterion equals to the posterior probability of the correct word sequence given a sequence of acoustic observations. We define the loss which we want to minimize as:

$$\begin{aligned} L_{MMI}(\theta) &= -\sum_{r=1}^{R} \log p_\theta(W_r|X_r) \\ &= -\sum_{r=1}^{R} \log \frac{p_\theta(X_r|W_r)^\kappa \cdot p(W_r)}{\sum_W p_\theta(X_r|W)^\kappa \cdot p(W)} \end{aligned} \quad (1)$$

where

$$p_\theta(X_r|W_r) = \max_{S_r} \prod_{t=1}^{T_r} p_\theta(x_{r,t}|s_{r,t}, W_r) \cdot p(s_{r,t}|s_{r,t-1}, W_r)$$

and

$$p(W_r) = \prod_{n=1}^{N_r} p(w_n|w_{n-m+1}^{n-1}).$$

In sequence training it is common to use an acoustic model scale κ instead of using language model scaling. We set $\kappa = \frac{1}{\lambda}$, where λ is the best language model scale from our baseline recognition step. The language model is an m-gram count model with Kneser-Ney smoothing. In the denominator of (1) we apply usual approximation using a lattice to approximate the summation.

The MPE criterion optimizes the average accuracy between competing word sequences and the reference one. Depending on the type of accuracy, MPE or sMBR can be defined. For MPE accuracy is defined on the phone level, for sMBR it is defined on the physical HMM states level. [3, 5]. The MPE loss is given by

$$L_{\text{MPE}}(\theta) = -\sum_{r=1}^{R} \frac{\sum_W p_\theta(X_r|W)^\kappa \cdot p(W) \cdot \mathcal{A}(W, W_r)}{\sum_{W'} p_\theta(X_r|W')^\kappa \cdot p(W')}.$$

4. PARALLELIZING THE LOSS COMPUTATION

For a given mini-batch z_s and parameters θ_s, the loss L can be written as a composed function

$$L(\theta_s, z_s) = L(y(\theta_s, z_s))$$

where y is the output of our model (here: neural network) and thus the gradient becomes

$$\nabla_\theta L(\theta_s, z_s) = \nabla_y L(y(\theta_s, z_s)) \cdot \nabla_\theta y(\theta_s, z_s).$$

For calculating the gradients to perform stochastic gradient descent (SGD), we will first calculate y and then L itself and then, following the back propagation algorithm, we firstly calculate $\nabla_y L(y(\theta_s, z_s))$ and then the remaining $\nabla_\theta y(\theta_s, z_s)$. Calculating y and L is usually called the forward propagation and calculating the gradients is the backward propagation of the error signals, where the top error signal is given by $\nabla_y L(y(\theta_s, z_s))$.

The idea is now to parallelize these calculations by dividing them onto three steps:

1. First forward propagation, i.e. calculation of y for a given θ_s and z_s. Effectively, we do that for multiple sequences in one mini-batch z_s.
2. Loss and error signal calculation, i.e. calculation of L and $\nabla_y L$. This will depend only on the previously calculated y but otherwise not directly on θ.
3. Second forward propagation to calculate y, then reusing the loss and top error signal from the previous step and continue with backward propagation with further model update θ.

The point of the second forward propagation is that θ will have changed compared to the first forward propagation and thus also all inner activations of the model. Hence, effectively, the only approximation comes from a delayed error signal. The SGD update rule can be written as

$$\theta_{s+1} = \theta_s - \gamma_s U_s$$

where γ_s is the learning rate in step s and U_s is the update. In the classic SGD

$$U_s = \nabla_\theta L(\theta_s, z_s).$$

In our case we have

$$\tilde{U_s} = \nabla_\theta L(\theta_{s-d}, z_s),$$

i.e. the gradient w.r.t. the model of d steps earlier.

Note that all this can be calculated in parallel. That is different from before, where forward propagation and loss calculation and backward propagation must be serial. In addition, loss calculation can also be done in parallel for multiple sequences at once, as well as the first forwarding and also the final forward and back propagation. As a result, when increasing d, we can parallelize the loss calculation as much as we want. Hence, we can decrease the total loss calculation time almost as much as we want until it plays a minor role of the total calculation time. One overhead from this method comes from the additional first forward propagation.

The final forward and back propagation is actually on frame-level because we have the error signal individually for every frame. We can further gain on runtime improvements by applying frame-level optimizations such as chunking, cf. Section 5.

4.1. Convergence proof sketch

We will provide a sketch for the proof of the convergence of this method. We will follow the proof of the convergence of stochastic gradient descent (SGD) by Léon Bottou in [18]. We want to find a minimum of $L(\theta) = \mathbb{E}_z L(\theta, z)$. We will only demonstrate the convex case here for simplification, i.e. we assume a single minimum θ^* of L. Also we assume that L satisfies the condition

$$\forall \epsilon > 0, \quad \inf_{(\theta-\theta^*)^2 > \epsilon} (\theta - \theta^*) \nabla_\theta L(\theta) > 0$$

which states that the opposite of the gradient $-\nabla_\theta L(\theta)$ always points towards the minimum θ^*. We show that the *Lyapunov process*

$$h_s = (\theta_s - \theta^*)^2$$

will converge to zero *almost surely*, i.e. with probability 1, denoted as $h_s \xrightarrow[s\to\infty]{\text{a.s.}} 0$. In the original proof, one would need the condition

$$\mathbb{E}_z U(\theta, z) = \nabla_\theta L(\theta).$$

Unfortunately, this condition does not hold for $\tilde{U}$. However, when we assume that there exists $A, B \geq 0$ such that

$$\frac{2}{\gamma_s} \cdot (\theta_s - \theta^*) \cdot (\nabla_y L(y(\theta_s, z_s)) - \nabla_y L(y(\theta_{s-d}, z_s))) \cdot \nabla_\theta y(\theta_s, z_s)$$
$$\leq A + B \cdot (\theta_s - \theta^*)^2 \quad (2)$$

and

$$\mathbb{E}\left(\tilde{U}(\theta, z)^2\right) \cdot 2 \leq A + B \cdot (\theta_s - \theta^*)^2$$

we can recover the proof given the same conditions on γ_s as in [18]. Note that the formula in (2) is likely bounded but we don't show that here. We will get

$$\mathbb{E}\left(h_{s+1} - (1 - \gamma_s^2 B) h_s\right) \leq \gamma_s^2 A$$

for some $A, B \geq 0$ and via the *quasi-martingale convergence theorem* it follows that h_s converges. Then it also follows that

$$(\theta_s - \theta^*) \nabla_\theta L(\theta_s) \xrightarrow[s \to \infty]{\text{a.s.}} 0$$

and thus $h_s \xrightarrow[s \to \infty]{\text{a.s.}} 0$ and $\theta_s \xrightarrow[s \to \infty]{\text{a.s.}} \theta^*$.

5. SEQUENCE SORTING AND FRAME CHUNKING

When we combine multiple full sequences in one mini-batch, the calculation will usually take as long as the longest sequence. If there is a huge variation in sequence lengths per mini-batch, this will have a big impact on the processing time. One way to overcome this is to sort the segments by length. To keep some randomness in the segment order, we randomly shuffle the segment order and then sort only batches of N segments with $N = 100$.

Another way to speed up the computation time for RNNs/LSTMs is to not train on the full sequence but use chunks of the sequence. This is only applicable in a straightforward way for frame-wise criteria and not for sequence criteria because you need to split up the targets in the same way. Consider a sequence with 1000 frames, then we could divide it into 10 chunks with 100 frames each. The processing of those 10 chunks will be faster than the calculation of the full sequence since we can calculate all chunks in parallel. The behavior of different chunking settings was studied in [19]. In Table 1 we can see that frame-wise training with frame chunking is more than 45% faster than training on the full sequences. Our parallel sequence training method will also allow us to use frame chunking in the second forward propagation and back propagation step, thus we can gain the same speed-up for this step. In [20] we described a way to combine the outputs of overlapping chunks to get back the full sequence. Then we could do sequence training in the usual way but we didn't investigate this possibility here.

6. IMPLEMENTATION AND CONFIGURATIONS

We use RASR [21, 22] to create the lattices and to calculate the loss and loss gradient for all criteria which are presented in this work. We use our Theano-based [23] framework RETURNN [24] to model the BLSTM and do control the overall training procedure. The parallel training method described in this paper is developed and published as part of RETURNN. We prepared all config files for performed experiments here [25].

Table 1: Training times for frame-wise trained systems, comparing sequence frame chunking (50:25) (cf. Section 5) with training using the full sequence, and additionally also sorting the sequences by length.

system	epoch time [secs]
chunking	**1205**
full seq	2331
full seq + sorting	2310

7. EXPERIMENTS

In this section we report experiments on the CHiME-3 speech recognition task [26]. The CHiME-3 scenario is ASR for a multi-microphone tablet device being used in everyday, noisy environments. The dataset contains WSJ0 speech recordings under different environment conditions such as public transport, pedestrian area, street and café.

A BLSTM supported GEV beamformer front-end was used as data processing and feature extraction [27]. A 2-gram LM is preferred for lattice generation and training in order to provide more variability for competing sequences in the lattice [6, 8]. and a 5-gram was used for recognition. The reported WERs are measured on the real-noisy development and evaluation data part from CHiME-3.

7.1. Baseline frame-wise trained system

In frame-wise and sequence training, our acoustic model is a BLSTM with 3 hidden layers with 500 hidden units in each direction, concatenated after every layer. We use dropout and L_2 regularization. Network optimization is done using mini-batch RMSProp [28] with the frame-wise cross-entropy loss function. The learning rate is controlled with some Newbob variant [24].

7.2. Sequence-discriminative trained system

Lattice-based sequence training requires a lattice (word graph) which approximates the search space. The word graph was created using model trained frame-wise. In this work we prepared several lattices using different pruning and LM history length, cf. Table 2.

For lattice generation language model scale 12.6 was used in all experiments. Prior distribution for states is computed during frame-wise training from the NN output layer as proposed in [30].

We checked the influence of LM history length on sMBR sequence training, cf. Table 4. Based on our experience on sequence training and results from Table 4, a weak language model is preferred to be used during lattice generation and training in order to provide more variability for the competing word sequences in the word graph [31]. In addition, lattice

Table 2: Properties of the lattices and impact of the language model history length. T-WER means WER on the training data. GER is Graph Error Rate (Oracle Word Error Rate) [29]. Density means lattice density which is a factor of the number of arcs in the lattice divided by the number of spoken words.

m-gram	T-WER [%]	GER [%]	density
2	5.42	1.09	106
	5.40	**0.59**	12388
4	**4.71**	1.40	33
	4.71	1.32	89

Table 3: Serial sequence training results with small lattices (1st row in Table 2)

system	criterion	WER [%]	
		dev	eval
frame-wise baseline	CE	6.49	8.43
serial seq. training	MMI	5.79	8.18
	MPE	5.53	8.16
	sMBR	**5.32**	**7.82**

with density in the range $100 - 1000$ and GER around 1% gives good improvements.

We performed sequence training with weak a language model using a number of sequence training criteria, cf. Table 3. Training with sMBR gave the best performance on the CHiME-3 task.

7.3. Parallel sequence training

We expect to reduce calculation time of the loss and back propagation. For serial training, to compute one mini-batch which involves copying the data to the GPU, forwarding, loss calculation and back propagation, the total time is 1.23 secs. on average with small lattices. The loss calculation takes 0.25 secs. (20%) and the back propagation 0.32 secs. (26%). For the bigger lattices, the loss calculation takes 25% of the mini-batch time.

In our experiments, we get a relative speedup of up to 24% still using one GPU and our parallel sequence training method (with chunking), cf. Table 5. Increasing the seq. delay d does not have a large impact on the WER, cf. Table 6. Interestingly, it seems to overfit more than the serial sequence training, and less so for bigger seq. delays. We did not increase the number of parallel threads in our experiments, although the extended possibility to parallelize the forwarding and loss calculation with a higher seq. delay d can in theory close the gap to get the full speed-up by chunking as in Table 1.

Table 4: Comparison between sMBR sequence training using different language model order (1st and 4th row of Table 2).

m-gram	WER [%]	
	dev	eval
2	**5.32**	**7.82**
4	5.53	8.07

Table 5: Training time comparison for sequence training with sMBR with small and big lattices (1st and 2nd rows of Table 2). In all experiments, the batch size is 5000. For parallel training, the chunk size is 100, seq. delay $d = 15$.

lattices	training	epoch time	WER [%]	
		[secs]	dev	eval
small	serial	449	**5.32**	**7.82**
	parallel	**343**	5.63	8.23
big	serial	453	5.39	8.01
	parallel	353	5.68	8.38

Table 6: Comparing seq. delays, parallel sMBR sequence training, with small lattice (1st row in Table 2).

seq. delay d	WER [%]		epoch time
	dev	eval	[secs]
15	**5.63**	8.23	343
30	5.72	7.95	342
100	5.75	**7.92**	341
1000	5.88	8.20	**332**

8. CONCLUSION & OUTLOOK

We presented a novel method which enables to parallelize some of the calculations during sequence training and thus allows us to run faster sequence training. It also allows to do apply frame-wise training methods like chunking to further improve the training time. In our experiments, we get a speedup of up to 26%, although we showed that there is still a lot of room for improvement. Note that Chime has quite short sequences — 644 time steps in average, so that the loss calculation is not so much a bottleneck for this case and we expect to get more speedup on tasks with longer sequences.

Also, we outlined how to further modify the method to increase the speed even more, e.g. by applying the method from [20] for the first forwarding pass.

In our knowledge most academic groups use FFNN-HMM sequence training and choose sMBR as a known best criterion. Here we explored sequence training with deep BLSTM networks and different criteria such as MMI, MPE and sMBR.

9. ACKNOWLEDGEMENTS

We thank Jahn Heymann, Lukas Drude, Aleksej Chinaev and Reinhold Haeb-Umbach for their CHiME-3 front-end which we used in this work. The research was partially supported by Deutsche Forschungsgemeinschaft (DFG) under Contract No. Schl2043/11-1 and additionally, the research was partially supported by Ford Motor Company.

10. REFERENCES

[1] Karel Veselỳ, Arnab Ghoshal, Lukás Burget, and Daniel Povey, "Sequence-discriminative training of deep neural networks.," in *INTERSPEECH*, 2013, pp. 2345–2349.

[2] Li Deng and Dong Yu, "Deep learning," *Signal Processing*, vol. 7, pp. 3–4, 2014.

[3] Daniel Povey, *Discriminative training for large vocabu-*

lary speech recognition, Ph.D. thesis, University of Cambridge, 2005.

[4] Ralf Schlüter, Wolfgang Macherey, Boris Müller, and Hermann Ney, "Comparison of discriminative training criteria and optimization methods for speech recognition," *Speech Communication*, vol. 34, no. 3, pp. 287–310, 2001.

[5] Brian Kingsbury, "Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2009, pp. 3761–3764.

[6] Georg Heigold, Hermann Ney, Ralph Schluter, and Simon Wiesler, "Discriminative training for automatic speech recognition: Modeling, criteria, optimization, implementation, and performance," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 58–69, 2012.

[7] Brian Kingsbury, Tara N Sainath, and Hagen Soltau, "Scalable minimum bayes risk training of deep neural network acoustic models using distributed hessian-free optimization," in *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[8] Simon Wiesler, Pavel Golik, Ralf Schlüter, and Hermann Ney, "Investigations on sequence training of neural networks," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 4565–4569.

[9] Hasim Sak, Oriol Vinyals, Georg Heigold, Andrew Senior, Erik McDermott, Rajat Monga, and Mark Mao, "Sequence discriminative distributed training of long short-term memory recurrent neural networks," in *Interspeech*, 2014.

[10] Zaihu Pang and Fengyun Zhu, "Noise-robust ASR for the third'chime'challenge exploiting time-frequency masking based multi-channel speech enhancement and recurrent neural network," *arXiv preprint arXiv:1509.07211*, vol. 1, 2015.

[11] Andrew Senior, Hasim Sak, Felix de Chaumont Quitry, Tara N. Sainath, and Kanishka Rao, "Acoustic modelling with CD-CTC-SMBR LSTM RNNS," in *ASRU*, 2015.

[12] Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahrmani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur, "Purely sequence-trained neural networks for asr based on lattice-free mmi," *To be published at Interspeech*, 2016.

[13] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 369–376.

[14] Haşim Sak, Andrew Senior, Kanishka Rao, and Françoise Beaufays, "Fast and accurate recurrent neural network acoustic models for speech recognition," *arXiv preprint arXiv:1507.06947*, 2015.

[15] Haşim Sak, Andrew Senior, Kanishka Rao, Ozan Irsoy, Alex Graves, Françoise Beaufays, and Johan Schalkwyk, "Learning acoustic frame labeling for speech recognition with recurrent neural networks," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 4280–4284.

[16] Tara Sainath, Kanishka Rao, et al., "Acoustic modelling with cd-ctc-smbr lstm rnns," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 2015, pp. 604–609.

[17] Georg Heigold, Erik McDermott, Vincent Vanhoucke, Andrew Senior, and Michiel Bacchiani, "Asynchronous stochastic optimization for sequence training of deep neural networks," in *ICASSP*, 2014.

[18] Léon Bottou, "Online learning and stochastic approximations," *On-line learning in neural networks*, vol. 17, no. 9, pp. 142, 1998.

[19] Albert Zeyer, Patrick Doetsch, Paul Voigtlaender, Ralf Schlüter, and Hermann Ney, "A comprehensive study of deep bidirectional LSTM RNNs for acoustic modeling in speech recognition," *arXiv preprint arXiv:1606.06871, submitted to ICASSP 2017*, 2016.

[20] Albert Zeyer, Ralf Schlüter, and Hermann Ney, "Towards online-recognition with deep bidirectional LSTM acoustic models," in *Interspeech*, 2016.

[21] Simon Wiesler, Alexander Richard, Pavel Golik, Ralf Schlüter, and Hermann Ney, "RASR/NN: The RWTH neural network toolkit for speech recognition," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Florence, Italy, May 2014, pp. 3313–3317.

[22] David Rybach, Stefan Hahn, Patrick Lehnen, David Nolden, Martin Sundermeyer, Zoltan Tüske, Simon Wiesler, Ralf Schlüter, and Hermann Ney, "RASR - the RWTH Aachen university open source speech recognition toolkit," in *IEEE Automatic Speech Recognition and Understanding Workshop*, Waikoloa, HI, USA, Dec. 2011.

[23] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.

[24] Patrick Doetsch, Albert Zeyer, Paul Voigtlaender, Ilya Kulikov, Ralf Schlüter, and Hermann Ney, "RETURNN: the RWTH extensible training framework for universal recurrent neural networks," *arXiv preprint arXiv:1608.00895, submitted to ICASSP 2017*, 2016.

[25] "GitHub repository with config files for experiments in RETURNN," 2016, `https://github.com/rwth-i6/returnn-experiments/tree/master/2016-seqtrain-paper`.

[26] Jon Barker, Ricard Marxer, Emmanuel Vincent, and Shinji Watanabe, "The third chime speech separation and recognition challenge: Dataset, task and baselines," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 2015, pp. 504–511.

[27] Jahn Heymann, Lukas Drude, Aleksej Chinaev, and Reinhold Haeb-Umbach, "BLSTM supported GEV beamformer front-end for the 3rd CHiME challenge," in *Automatic Speech Recognition and Understanding Workshop (ASRU 2015)*, December 2015.

[28] Tijmen Tieleman and Geoffrey Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning*, vol. 4, no. 2, 2012.

[29] Gunnar Evermann, "Minimum word error rate decoding," MPhil thesis, Cambridge University, UK, 1999.

[30] Vimal Manohar, Daniel Povey, and Sanjeev Khudanpur, "Semi-supervised maximum mutual information training of deep neural network acoustic models," in *Proceedings of INTERSPEECH*, 2015.

[31] R. Schlüter, B. Müller, F. Wessel, and H. Ney, "Interdependence of language models and discriminative training," in *IEEE Automatic Speech Recognition and Understanding Workshop*, Keystone, CO, Dec. 1999, vol. 1, pp. 119–122.