Purely sequence-trained neural networks for ASR based on lattice-free MMI

Daniel Povey[1,2], *Vijayaditya Peddinti*[1], *Daniel Galvez*[3], *Pegah Ghahrmani*[1], *Vimal Manohar*[1], *Xingyu Na*[4], *Yiming Wang*[1], *Sanjeev Khudanpur*[1,2]

[1]Center for Language and Speech Processing
[2]Human Language Technology Center of Excellence
The Johns Hopkins University, Baltimore, MD 21218, USA

[3]Department of Computer Science, Cornell University
[4]Lele Innovation and Intelligence Technology (Beijing) Co., Ltd.

{dpovey,dt.galvez,asr.naxingyu}@gmail.com, {vijay.p,vmanoha1,pghahre1,yiming.wang,khudanpur}@jhu.edu

Abstract

In this paper we describe a method to perform sequence-discriminative training of neural network acoustic models without the need for frame-level cross-entropy pre-training. We use the lattice-free version of the maximum mutual information (MMI) criterion: LF-MMI. To make its computation feasible we use a phone n-gram language model, in place of the word language model. To further reduce its space and time complexity we compute the objective function using neural network outputs at one third the standard frame rate. These changes enable us to perform the computation for the forward-backward algorithm on GPUs. Further the reduced output frame-rate also provides a significant speed-up during decoding.

We present results on 5 different LVCSR tasks with training data ranging from 100 to 2100 hours. Models trained with LF-MMI provide a relative word error rate reduction of $\sim$11.5%, over those trained with cross-entropy objective function, and $\sim$8%, over those trained with cross-entropy and sMBR objective functions. A further reduction of $\sim$2.5%, relative, can be obtained by fine tuning these models with the word-lattice based sMBR objective function.

Index Terms: neural networks, sequence discriminative training

1. Introduction

Sequence discriminative training of neural networks for Automatic Speech Recognition (ASR) has been shown to provide significant reduction in word error rates (WERs), vs. cross-entropy training ([1, 2, 3]). Traditionally this has been done by training a cross-entropy system, generating word lattices with a weak language model, and using these lattices as an approximation for all possible word sequences in the discriminative objective function– as was done when Gaussian Mixture Models were the state of the art [4].

Recently, Connectionist Temporal Classification (CTC) [5] has attracted a lot of attention in speech recognition applications [6, 7], although the only reports of its success have been with huge amounts of data. Our attempts to get CTC to beat cross-entropy trained systems on mere hundreds of hours of data were unsuccessful– and beyond the scope of this paper– but we realized that some ideas recently applied with CTC could be used in the context of MMI-based sequence training. Like MMI, CTC training maximizes the conditional log-likelihood of the correct transcript; the difference is that in CTC the probabilities are locally normalized but in MMI they are globally normalized.

The ideas that we apply to MMI-based training are:

- Training from scratch without initialization from a cross-entropy system
- The use of a 3-fold reduced frame rate [8] (and a simpler HMM topology)
- Limiting the range of time frames where supervision labels can appear by using Finite State Acceptors [9]

Because our method is denominator-lattice-free, we do the summation over all possible label sequences on the GPU; to keep this manageable, we use a phone-level, not word-level, language model. Overfitting is a problem; we use a combination of three different regularization techniques.

In Section 2 we discuss various details of the proposed method. In Section 3 we describe the experimental setup and present the results in Section 4. Finally we present our conclusions in Section 5.

2. Proposed Method

The basic premise of this paper is to do MMI training directly on the GPU, without lattices, using the forward-backward algorithm for both numerator and denominator parts of the objective function. Obviously efficiency is going to be a problem [10]. Because fast GPU-based computations benefit from synchronized memory access across cores, we avoid pruned versions of forward-backward, and instead focus on making the graph as small as possible.

We don't give any equations here, because MMI training is well known (e.g. see [4]). We interpret the neural net output as the log of a pseudo-likelihood; there is no acoustic scaling factor; and there is no division by the prior.

The computation of the derivatives of the objective function requires us to compute two sets of posterior quantities: one from the *numerator graph*, which is specific to each utterance and which encodes the supervision information; and one for the *denominator graph*, which encodes all possible word sequences and which is the same for all utterances. Both of these can be viewed as equivalent to HMMs, although we use the Finite State Acceptor (FSA) format to store both of them (with labels on arcs, not states).

The authors would like to thank Xiaohui Zhang for his help with the *nnet3* framework, and Tony Robinson, Rémi Francis, Charles Corfield and Mike Newman for comments on the paper draft. This work was partially supported by NSF CRI Grant No 1513128, DARPA LORELEI Contract No HR0011-15-2-0024 and IARPA BABEL Contract No 2012-12050800010, a faculty gift from Facebook, and a research award from Google.

2.1. Topology and decision trees

The conventional HMM topology in ASR is a 3-state left-to-right HMM that can be traversed in a minimum of 3 frames. Here, we use a topology that can be traversed in one frame. After comparing various topologies, we settled on a topology where the first frame of a phone has a different label than the remaining frames (a different pdf-id, in Kaldi terminology, i.e. it maps to a different output of the neural net), so a single HMM may emit either a, or ab, or abb, etc. The reader is free to consider the b as analogous to the blank symbol in CTC (while bearing in mind that in general each triphone may get its own version of the b symbol).

We build the phonetic-context decision tree specifically for this topology and frame rate after converting alignments from a traditional HMM-GMM system at the normal frame rate; the decision-tree is then built using the same procedure and the same features (MFCC+LDA+MLLT) as for our HMM-GMM system. The optimal number of leaves tends to be a little smaller than than for a cross-entropy neural network.

2.2. Transition modeling

In our baseline cross-entropy based HMM-DNN framework, the HMMs use transition probabilities; these are estimated in the conventional way for HMMs. In this work we just set the transition probabilities to be a constant value (0.5) that makes each HMM-state sum to one. For the topologies we use, estimating the transition probabilities would add no modeling power anyway (depending on the exact granularity with which they are shared).

2.3. Denominator language model

To create the denominator graph we need a language model. In traditional MMI [4], a unigram or sometimes bigram word-level language model is used. However, this would be too slow in our scenario. Instead we use a 4-gram phone level language model, estimated from phone-level alignments of the training data. It is constructed in a special way that is designed to minimize the size of the denominator graph. At and below the trigram level there is no smoothing or pruning; it can only predict triphones that were seen in the training data. This limits how much the addition of phonetic context-dependency can increase the size of the graph. In addition to the un-pruned trigram language model, we select a specified number of 4-gram history states (2000 in experiments reported here), selected so as to maximize the likelihood of the training data. There is no interpolation or backoff from history states that exist, e.g. if a 3-phone history is available we do not interpolate with, or back off to, the 2-phone history (so test-data perplexity is infinite).

2.4. Denominator graph creation

The construction of the denominator graph from the phone-level denominator LM can be summarized as follows (see [11] for notation). First we compose on the left with C to add phonetic context-dependency. Then we compose on the left with H (and project on the input) to add the correct HMM topology and to replace the labels with indexes of context-dependent states; and we remove epsilons. At this point the graph is usable, but we want to reduce the number of states and transitions as much as possible for efficiency. We found that the standard technique based on determinization (with disambiguation symbols) and minimization was not effective. Instead we perform the sequence (push the weights; minimize; reverse the FSA; push the weights; minimize; reverse) 3 times, and remove ϵs again (since reversal adds a few). Most of the minimization happens on the first one or two iterations. Note: the minimization algorithm of Hopcroft ([12]) is applicable to non-deterministic FSAs.

In the Switchboard-1 system described in the experimental section, with 7115 leaves, the final compiled graph had about 24k states and 220k transitions. Both are important to us because the number of states dictates memory consumption and the number of transitions dictates the time taken by the denominator forward-backward. In the setups described here the denominator forward-backward took less than 20% of the time in the training program, with the rest consisting of neural network computation.

2.5. Initial and final probabilities in the denominator graph

The denominator graph as created above has an initial state and final probabilities that reflect the statistics of sentence starts and ends. This is incompatible with the fact that we train on fixed size chunks extracted from utterances (typically 1.5 seconds long). So we use modified initial and final probabilities in the forward-backward computation. The initial probabilities are obtained by running the HMM for 100 time steps starting from the initial state, and averaging the distribution of states over those 100 time steps. The final probabilities are all set to one. For convenience in creating the numerator graphs (see Sec. 2.6), we create something that we call the *normalization FSA* that is the same as the denominator FSA but with these modified initial and final probabilities (the initial probabilities are represented by adding ϵ transitions from a new initial state, and then removing epsilons).

2.6. Numerator graph creation

2.6.1. Time constraints

The process of numerator graph creation for an utterance begins with the same process by which we create an utterance-specific Finite State Transducer (FST) for Viterbi alignment during conventional training. (Note: here, our numerator and denominator graphs are actually FSAs, and we will use that term from now on). If we were training on entire utterances rather than fixed-sized chunks, we could use this directly. However, in order to make it possible to split up the numerator graphs appropriately, we need to add time constraints to the alignments.

Prior to training the neural net, we use a GMM-based system to dump lattices representing alternative pronunciations of the training utterances. (We use a modified version of the determinization procedure of [13] in which the phone sequence as well as the word sequence is encoded in the output symbols, so distinct pronunciations are retained). These lattices are processed into phone graphs and then compiled into utterance-specific FSAs as for conventional training. Separately the lattices are also processed into a frame-by-frame mask of what phones are allowed to appear on what frames: a user-specifiable tolerance (50 milliseconds by default) allows a phone to appear slightly before or after where it appeared in the lattice. We obtain the mask at the subsampled frame rate i.e., with a 30ms frame shift. Suppose the number of subsampled frames is T. We compose the previously created decoding graph with an FSA that has T+1 states, with the first state initial and the last state final, and has a transition from state t to $t+1$ with a particular pdf-id on it, if that pdf-id is from a phone that is allowed on the t'th frame. This allows us to split up the numerator FSA into appropriate chunks, because each state of the composed FSA

can now be identified with a frame index. A similar idea was used in [9] for CTC training.

2.6.2. Adding weights to the numerator FSA

For diagnostic reasons it is convenient to have an objective function (expressed as a log-prob per frame) that can never be greater than zero. We achieve this by adding the weights from the denominator FSA into the numerator FSA; this includes the modified initial and final probabilities discussed in Section 2.5. This is done by composing the numerator FSA with the *normalization FSA* mentioned above. While creating the numerator FSA we avoid adding in transition probabilities (i.e. they are all set to 1.0), to prevent counting them twice. At the end, the numerator FSA can be thought of as containing a subset of the paths in the denominator FSA.

2.6.3. Forward-backward computation

The forward-backward computation needed to compute the derivatives *w.r.t.* the neural net output is implemented on the CPU for the numerator, which is simpler. The denominator forward-backward computation is done on the GPU. In order to avoid the overhead of log and exp operations, we don't do the computation in log-space. This requires some care to avoid underflow or overflow. On each frame we multiply the neural net outputs by a value chosen to keep the α and β quantities within a reasonable range (we use the inverse of the sum of the previous frame's αs). The "leaky-hmm" idea discussed below is helpful for this, as it ensures that if the α values are kept in a reasonable range, the β values cannot overflow.

2.7. Regularization

Sequence level training tends to overfit [3]. We combine three different regularization methods to reduce this.

2.7.1. Cross-entropy regularization

We add to the network a separate output layer specifically for training with the cross entropy objective function; and we also give it its own version of the last hidden layer, so two of the weight matrices are specific to the cross-entropy training. The supervision for the cross-entropy objective is a "soft" supervision derived from the posteriors in the numerator forward-backward computation; this is for convenience. We scale the cross-entropy objective by a user-specified constant (normally 0.1) during training, because its dynamic range per frame is naturally greater than that of the MMI objective. The parameters specific to the cross-entropy output may be discarded after training.

2.7.2. Output l_2-norm regularization

We penalize the squared l_2 norm of the output of the neural network (the main output, not the cross-entropy one). This consists of adding $-0.5c\ \mathbf{y} \cdot \mathbf{y}$ to the objective function for each output frame, where $\mathbf{v}$ is the neural network output on that frame. We set c to 0.0005.

2.7.3. Leaky HMM

We allow for transition probabilities from each state in the HMM to every other state, to ensure gradual forgetting of context. It's equivalent to stopping and restarting the HMM with some probability on each frame. It can be described as an ϵ transition from each state a to each other state b with probability equal to *leaky-hmm-coefficient* times the initial-probability of state b (except that we only get to traverse one of these ϵ transitions per frame). For further details see the code[1]. Here we set *leaky-hmm-coefficient* is set to 0.1.

3. Experimental Setup

We report results on several different LVCSR tasks, however most results are presented on 300 hour subset of the Switchboard task (SWBD-300Hr). The experimental setup is similar to [14] and is described here briefly. The experimental setups for other LVCSR tasks are similar and details are available in the code repository [15].

The HMM-GMM system, used to generate the alignments for cross-entropy training of neural network and the numerator lattices for LF-MMI training (see 2.6); and the language model used for constructing the decode graph are similar to those described in [2]. We use the *speed-perturbation* technique ([16]) for 3-fold data augmentation; and iVectors to perform *instantaneous adaptation* of the neural network ([17]). We use an enhanced lexicon FST for decoding test utterances, with probabilities for pronunciations of each word and also encodes the probability of silence before and after each pronunciation explicitly ([18]). WER results are reported after 4-gram LM rescoring of lattices generated using a trigram LM.

3.1. Neural networks

The acoustic models used are sub-sampled time-delay neural networks (TDNNs, [14]), long short term memory networks (LSTMs, [19]) and bidirectional LSTMs (BLSTMs).

3.1.1. Time delay neural networks

The TDNNs used for cross-entropy training are similar to those specified in [14], except with a slightly different configuration of splicing indexes. The splicing indexes used are `-1,0,1 -1,0,1,2 -3,0,3 -3,0,3 -3,0,3 -6,-3,0 0`. The initial `-1,0,1` means that the first layer sees 3 consecutive frames of input; the `-3,0,3` means that most hidden layers see 3 frames of the previous layer, separated by 3 frames. Since these differ by multiples of 3 and we only evaluate the output at multiples of 3 frames, most hidden layers only need to be evaluated every 3 frames, like the output, which is efficient. The final `0` means that the very last weight matrix does not have spliced input. Our chosen ReLU dimensions (the output dimensions of the weight matrices) tend to be smaller than our cross-entropy trained TDNNs, e.g. 576 for the Switchboard-1 system, versus 1024 for the cross-entropy trained baseline. The cross-entropy baseline for that system had splicing indexes `-2,-1,0,1,2 -1,2 -3,3 -7,2 0`.

3.1.2. (B)LSTM networks

We use (B)LSTM layers with recurrent and non-recurrent projections as suggested in [19]. LSTMs used in cross-entropy training have a different delay at each layer, which increases as we go deeper into the network. This is done to reduce the computation. We use delays of -1, -2 and -3 at layers 1, 2 and 3 respectively. Similarly the BLSTM has increasing delays

[1] All of this is in published Kaldi code at `github.com/kaldi-asr/kaldi`; see the `chain` directory, and search for 'leaky' to find code related to this topic

along both directions. The (B)LSTM networks trained with LF-MMI have the same architecture as above, except for the delays. These networks use a constant delay of -3 (or +3) at each layer. This ensures that the amount of computation needed for generating outputs at a reduced frame rate of 33 Hz is one third of that required for computing outputs at 100 Hz.

4. Results

Table 1 shows a comparison of different regularization methods used in LF-MMI. All three provided gains individually (with *leaky-hmm* making less difference than the others). The gains were largely additive. LF-MMI is used with all the three regularization functions in the remainder of the experiments.

Table 1: Comparison of regularization functions with TDNN-B on the Hub5 '00 eval set, using SWBD-300 Hr data

Regularization Function			WER	
Cross-entropy	output l_2-norm	leaky HMM	Total	SWBD
N	N	N	16.8	11.1
Y	N	N	15.9	10.5
N	Y	N	15.9	10.4
N	N	Y	16.4	10.9
Y	Y	N	15.7	10.3
Y	N	Y	15.7	10.3
N	Y	Y	15.8	10.4
Y	Y	Y	15.6	10.4

Table 2 shows a comparison of the training methods. Comparing rows 1-3 it can be seen that LF-MMI training leads to WER improvements relative to both the cross-entropy (CE) and cross-entropy pretrained sMBR (CE $\rightarrow$ sMBR) objectives. Further the word LM based sMBR objective can provide gains even over LF-MMI trained models.

TDNN-A has the same configuration as the one described in [14]. The parameters in TDNN-A were reduced when training it with LF-MMI objective. TDNN-B and TDNN-C are uniformly sampled TDNNs (see 3.1.1) with larger number of layers than TDNN-A.

Table 2: Comparison of objective functions on Hub5 '00 eval set, using SWBD-300 Hr data

Objective function	Model (Size)	WER	
		Total	SWBD
CE	TDNN-A (16.6 M)	18.2	12.5
CE $\rightarrow$ sMBR	TDNN-A (16.6 M)	16.9	11.4
LF-MMI	TDNN-A (9.8 M)	16.1	10.7
	TDNN-B (9.9 M)	15.6	10.4
	TDNN-C (11.2 M)	15.5	10.2
LF-MMI $\rightarrow$ sMBR	TDNN-C (11.2 M)	15.1	10

Table 3 shows the performance gains due to LF-MMI across 3 models. It can be seen that the relative gains are smaller with the RNN architectures, compared to the feed-forward DNN architecture. The (B)LSTM network has 3 layers each with a cell dimension of 1024; and recurrent and non-recurrent projections of size 256 units.

Table 4 presents comparison of CE, CE $\rightarrow$ sMBR and LF-MMI across datasets of different sizes. It can be see that the gains are consistent across most datasets. We are currently investigating the relative increase in WER in the TED-LIUM task ([20]).

Table 3: Performance of LF-MMI with different models on the Hub5 '00 eval set, using SWBD-300 Hr data

Model	WER	
	Total	SWBD
TDNN-C + CE	18.2	12.5
TDNN-C + LF-MMI	15.5	10.2
LSTM + CE	16.5	11.6
LSTM + LF-MMI	15.6	10.3
BLSTM + CE	14.9	10.3
BLSTM + LF-MMI	14.5	9.6

Table 4: Performance of LF-MMI on various LVCSR tasks with different amount of training data, using TDNN acoustic models

Database	Size	WER		
		CE	CE $\rightarrow$ sMBR	LF-MMI
AMI-IHM	80 hrs	25.1	23.8	22.4†
AMI-SDM	80 hrs	50.9	48.9	46.1†
TED-LIUM	118 hrs	12.1	11.3	11.2*
Switchboard	300 hrs	18.2	16.9	15.5
Librispeech	1000 hrs	4.97	4.56	4.28
Fisher + SWBD	2100 hrs	15.4	14.5	13.3

†: In AMI LVCSR task adding a data filtering stage to eliminate large oracle-WER utterances increased the relative gain [21]
*: We used a data cleanup stage to remove parts of training recordings that are mistranscribed or do not correpsond to the marked speaker.
Note: All these experiments use the same TDNN architecture, except for the hidden layer size

Table 5 compares the results in this paper with those from [22] and [23] on the Hub5 '00 and RT03S eval sets.

Table 5: Comparison with systems in [22] and [23]

System	AM Dataset	LM Dataset	Hub5 '00		RT03S	
			SWB	CHM	FSH	SWB
Mohd. *et al* [22]	F+S	F+S	10.6	–	13.2	18.9
Mohd. *et al* [22]	F+S	F+S+O	9.9	–	12.3	17.8
Mohd. *et al* [22]	F+S+O	F+S+O	9.2	–	11.5	16.7
Saon *et al* [23]	F+S+C	F+S+O	8.0	14.1	–	–
TDNN+LF-MMI	S	F + S	10.2	20.5	14.2	23.5
TDNN+(LF-MMI$\rightarrow$sMBR)	S	F + S	10.0	20.1	13.8	22.1
BLSTM+LF-MMI	S	F + S	9.6	19.3	13.2	20.8
TDNN+LF-MMI	F + S	F + S	9.2	17.3	9.8	14.8
BLSTM+LF-MMI	F + S	F + S	8.5	15.3	9.6	13.0

F : Fisher corpus S : Switchboard corpus
C : Callhome corpus O : Other corpora

5. Conclusion and Future work

In this paper we applied ideas from recent CTC efforts [5, 6, 7, 9, 8] to MMI sequence discriminative training starting from randomly initialized neural networks. We showed that using a denominator-lattice-free version of MMI training, and a reduced frame rate, we were able to get $\sim 8\%$ relative improvement compared to the sMBR objective (with cross-entropy pretraining) and $\sim 11.5\%$ relative improvement compared to cross-entropy training.

We showed that the gains were consistent across most datasets and model types. We are currently investigating the sensitivity of this method to transcription errors (initial results show that it is more sensitive); and the difference in performance across feed-forward and recurrent neural networks.

6. References

[1] B. Kingsbury, "Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling," in *Proceedings of ICASSP*. IEEE, apr 2009, pp. 3761–3764.

[2] K. Veselỳ, A. Ghoshal, L. Burget, and D. Povey, "Sequence-discriminative training of deep neural networks." in *Proceedings of INTERSPEECH*, 2013, pp. 2345–2349.

[3] H. Su, G. Li, D. Yu, and F. Seide, "Error back propagation for sequence training of Context-Dependent Deep Networks for conversational speech transcription," in *Proceedings of ICASSP*. IEEE, may 2013, pp. 6664–6668.

[4] D. Povey, "Discriminative training for large vocabulary speech recognition," Ph.D. dissertation, Cambridge University, 2005.

[5] A. Graves, S. Fernández, F. Gomez, and J. 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.

[6] H. Sak, A. Senior, K. Rao, O. Irsoy, A. Graves, F. Beaufays, and J. Schalkwyk, "Learning acoustic frame labeling for speech recognition with recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4280–4284.

[7] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates *et al.*, "Deep speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[8] H. Sak, A. Senior, K. Rao, and F. Beaufays, "Fast and accurate recurrent neural network acoustic models for speech recognition," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[9] A. Senior, H. Sak, F. de Chaumont Quitry, T. N. Sainath, and K. Rao, "Acoustic Modelling with CD-CTC-SMBR LSTM RNNS," in *ASRU*, 2015.

[10] S. Chen, B. Kingsbury, Lidia Mangu, D. Povey, G. Saon, H. Soltau, and G. Zweig, "Advances in speech transcription at IBM under the DARPA EARS program," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 14, no. 5, pp. 1596–1608, sep 2006.

[11] M. Mohri, F. Pereira, and M. Riley, "Speech recognition with weighted finite-state transducers," in *Springer Handbook of Speech Processing*. Springer, 2008, pp. 559–584.

[12] J. Hopcroft, "An n log n algorithm for minimizing states in a finite automaton," *Theory of Machines and Computations*, pp. 189–196, 1971.

[13] D. Povey, M. Hannemann, G. Boulianne, L. Burget, A. Ghoshal, M. Janda, M. Karafiát, S. Kombrink, P. Motlicek, Y. Qian *et al.*, "Generating exact lattices in the wfst framework," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2012, pp. 4213–4216.

[14] V. Peddinti, D. Povey, and S. Khudanpur, "A time delay neural network architecture for efficient modeling of long temporal contexts," in *Proceedings of INTERSPEECH*, 2015.

[15] (2016) Readme file in code repository to replicate the experiments in this paper. [Online]. Available: https://github.com/vijayaditya/kaldi/blob/chain_paper_results/egs/README_paper_results

[16] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur, "Audio augmentation for speech recognition," in *Proceedings of INTERSPEECH*, 2015.

[17] G. Saon, H. Soltau, D. Nahamoo, and M. Picheny, "Speaker adaptation of neural network acoustic models using i-vectors," in *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, Dec 2013, pp. 55–59.

[18] G. Chen, H. Xu, M. Wu, D. Povey, and S. Khudanpur, "Pronunciation and silence probability modeling for ASR," in *Proceedings of INTERSPEECH*, 2015.

[19] H. Sak, A. Senior, and F. Beaufays, "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition," Feb. 2014. [Online]. Available: http://arxiv.org/abs/1402.1128

[20] A. Rousseau, P. Deléglise, and Y. Estève, "Ted-lium: an automatic speech recognition dedicated corpus." in *LREC*, 2012, pp. 125–129.

[21] V. Peddinti, V. Manohar, Y. Wang, D. Povey, and S. Khudanpur, "Far-field ASR without parallel data," in *Proceedings of Interspeech*, 2016. [Online]. Available: http://www.danielpovey.com/files/2016_interspeech_ami.pdf

[22] A.-r. Mohamed, F. Seide, D. Yu, J. Droppo, A. Stolcke, G. Zweig, and G. Penn, "Deep bi-directional recurrent networks over spectral windows," in *Proceedings of ASRU*. ASRU, 2015.

[23] G. Saon, H.-K. J. Kuo, S. Rennie, and M. Picheny, "The IBM 2015 English Conversational Telephone Speech Recognition System," may 2015. [Online]. Available: http://arxiv.org/abs/1505.05899