JHU ASPIRE SYSTEM : ROBUST LVCSR WITH TDNNS, IVECTOR ADAPTATION AND RNN-LMS

Vijayaditya Peddinti[1], *Guoguo Chen*[1], *Vimal Manohar* [1], *Tom Ko* [3] *Daniel Povey*[1,2], *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]Huawei Noah's Ark Research Lab, Hong Kong, China
{vijay.p,guoguo,khudanpur}@jhu.edu, {vimal.manohar91, tomkocse, dpovey}@gmail.com

ABSTRACT

Multi-style training, using data which emulates a variety of possible test scenarios, is a popular approach towards robust acoustic modeling. However acoustic models capable of exploiting large amounts of training data in a comparatively short amount of training time are essential. In this paper we tackle the problem of reverberant speech recognition using 5500 hours of simulated reverberant data. We use time-delay neural network (TDNN) architecture, which is capable of tackling long-term interactions between speech and corrupting sources in reverberant environments. By sub-sampling the outputs at TDNN layers across time steps, training time is substantially reduced. Combining this with distributed-optimization we show that the TDNN can be trained in 3 days using up to 32 GPUs. Further, iVectors are used as an input to the neural network to perform instantaneous speaker and environment adaptation. Finally, recurrent neural network language models are applied to the lattices to further improve the performance. Our system is shown to provide state-of-the-art results in the IARPA ASpIRE challenge, with 26.5% WER on the dev_test set.

Index Terms: far field speech recognition, time delay neural networks, iVectors, recurrent neural network language models

1. INTRODUCTION

Reverberant speech is assumed to be composed of direct-path response, early reflections and late reverberations. Early reflections, *viz.,* reflections within a delay of 50ms of the direct signal, can be effectively dealt with using DNN architectures which operate on comparatively short temporal contexts. However in order to tackle late reverberations, with *reverberation time* from 200 to 1000 ms in typical office environments [1], DNNs should be able to model temporal relationships across wide acoustic contexts.

In this paper we use a time delay neural network [2], which is a feed forward network architecture that is effective in modelling long term temporal contexts. In [3] it was shown that TDNNs can be trained with training times competitive with those of standard feed-forward DNNs, by sub-sampling the TDNN layer outputs. In this paper we use the TDNN architecture suggested in [3] for learning reverberation robust representations. The TDNN was able to benefit from increasing the input context up to 280 milliseconds. The ability to process such a wide temporal context enables the network to deal with late reverberations.

iVectors which capture both speaker and environment specific information have been shown to be useful for rapid adaptation of the neural network [4, 5, 6]. iVector based adaptation has also been shown to be effective in reverberant environments [7]. In this paper we use this adaptation technique.

We show experimental results on the ASpIRE far-field speech recognition challenge held by IARPA [8]. This challenge uses the English portion of the Fisher database [9] for acoustic and language model training. We show that in this large data scenario the proposed network architecture, combined with a distributed optimization technique [10], can train on multi-condition training data of $\sim$ 5500 hours, using up to 32 GPUs, in 3 days.

Using the TDNN architecture helps us to achieve results close to those of the best combined system submitted to the ASpIRE challenge, while using only a single system. Our system was able to achieve 26.5% WER on the *dev-test* set, while the next best system achieved 27.2% WER [1].

The paper is organized as follows, Section 2 describes the acoustic model, Section 3 describes the language model, Section 4 analyses the results, and conclusions are presented in Section 5.

2. ACOUSTIC MODEL

2.1. Input Features

Mel-frequency cepstral coefficients (MFCCs) [11], without cepstral truncation, were used as input to the neural network i.e., 40 MFCCs were computed at each time step. Readers are recommended to see [10] for a more detailed discussion on input data representation used here and its comparison with log mel features. MFCCs over a wide asymmetric temporal context were provided to the neural network. Different contexts were explored in this paper.

In this paper we use a iVector adapted neural network acoustic model. On each frame we append a 100-dimensional iVector [12] to the 40-dimensional MFCC input. The MFCC input is not subject to cepstral mean normalization; the intention is to allow the iVector to supply the information about any mean offset of the speaker's data, so the network itself can do any feature normalization that is needed. In order for the mean-offset information to be encoded in the iVector, we estimate the iVector on top of features that have not been mean-normalized. However, the Gaussian posteriors used for the iVector estimation are based on features that have been mean normalized using a sliding window of 6 seconds.

This work was partially supported by NSF Grants No IIA 0530118 and IIS 0963898, and DARPA BOLT Contract No HR0011-12-C-0015.

[1] At the end of the evaluation our system had 27.7% WER on dev-test

2.2. Neural Network Architecture

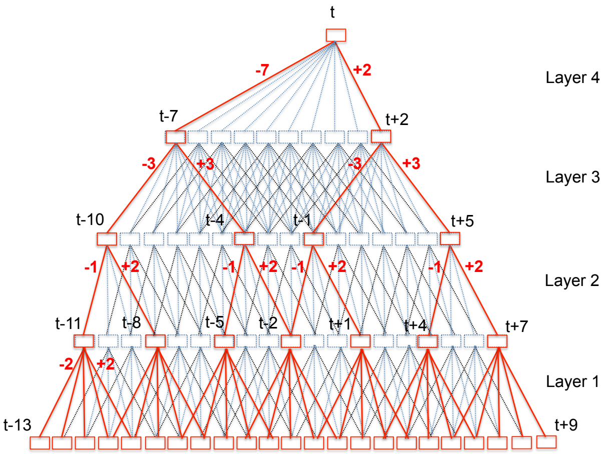

Fig. 1: Computation in TDNN with sub-sampling (red) and without sub-sampling (blue+red)

In this paper we use the sub-sampled TDNN architecture which was shown to be effective for modelling long term temporal dependencies in close-talk microphone speech, telephone speech and reverberant speech [3, 13].

The transforms in the TDNN architecture are tied across time steps and for this reason they are seen as a precursor to the convolutional neural networks. During back-propagation, due to tying, the lower layers of the network are updated by a gradient accumulated over all the time steps of the input temporal context. Thus the lower layers of the network are forced to learn translation invariant feature transforms [2].

The hyper-parameters which define the TDNN network are the input contexts at each layer required to compute an output activation, at one time step. A sample TDNN network is shown in Figure 1. The figure shows the time steps at which activations are computed, at each layer, and dependencies between activations across layers. It can be seen that the dependencies across layers are localized in time.

2.2.1. *Sub-sampling*

In a normal TDNN all the hidden activations in the input context are all spliced together into a larger vector, which goes through an affine transformation. This process is then repeated to compute hidden activations at all time-steps. However there are large overlaps between input contexts of activations computed at neighboring time steps. Under the assumption that neighboring activations are correlated, they can be sub-sampled.

In the sub-sampled TDNN, we select non-contiguous temporal frames, at each layer. In fact, in the hidden layers of the network, we generally splice no more than two frames to compute hidden activations at a time step.

Empirically we identified that splicing together activations separated by increasingly wide context, as we go to higher layers of the network leads to better results. The configuration in Figure 1, which is fairly typical, splices together frames $t-2$ through $t+2$ at the input layer (which we could write as context $\{-2,-1,0,1,2\}$ or more compactly as $[-2,2]$); and then at three hidden layers we splice frames at offsets $\{-1,2\}$, $\{-3,3\}$ and $\{-7,2\}$[2].

With the current sub-sampling scheme the overall necessary computation is reduced during the forward pass and backpropagation, due to selective computation of time steps. The training time of TDNN in Figure 1, without sub-sampling, is ∼10x compared to that of DNN with same number of layers. With proposed sub-sampling it is ∼2x the training time of DNN. Thus the sub-sampling process speeds up the TDNN training by ∼5x. Another advantage of using sub-sampling is the reduction in the model size. Splicing contiguous frames at hidden layers would require us to either have a very large number of parameters, or reduce the hidden-layer size significantly.

We use asymmetric input contexts, with more context to the left, as this reduces the latency of the neural network in online decoding, and also because this seems to be more optimal from a WER perspective.

A major difference in the current architecture compared to [2] is the use of the p-norm nonlinearity [14], which is a dimension reducing non-linearity. p-norm units with group size of 10 and $p = 2$ were used across all neural networks in our experiments, based on the observations made in [14].

2.3. Data Augmentation

Speech from the English portion of the Fisher corpus [9] (LDC2004S13, LDC2005S13) was used to train the acoustic models. Multi-condition training data was created by distorting speech with real world room impulse response (RIR) and noise recordings available from three different databases *viz.,* the RWCP sound scene database [3] [15], the REVERB challenge database [16] and the Aachen impulse response database [17]. 325 muti-channel recordings of RIRs were selected from the three databases. Noise recordings, containing stationary background noise caused mainly by air conditioning systems in a room and measured with the same microphone arrays as used for RIR measurement were available for 51 RIRs. The first channel from the multi-channel recordings of noise or RIR, was used for corruption.

Three different copies of each recording in the Fisher corpus were created by randomly sampling three different RIRs. When available the noise recordings were added to the RIR convolved speech data to have resultant SNR of 20, 15, 10, 5 and 0 dB.

Overall ∼ 5500 hours of training data was created, based on these copies. Another version of the acoustic model was trained on data that was processed as above, but also then speed perturbed as in [18]. We still produced 3 copies of each original recording, by combining a random RIR with each different speed of the data. Speed perturbation, which emulates pitch and tempo variations in speech, was shown to provide on average 4.3% relative gain in a variety of LVCSR tasks. However this did not help in the current task, possibly because we already had enough training-data variation from the reverberation and noise.

2.3.1. *Volume Perturbation*

The iVector based TDNN system relies on the neural network to learn the necessary normalization, based on mean shifts captured in the iVector. However in well curated audio databases there is low variance in audio volume, leading to low variance in iVector *w.r.t.* mean shifts. Performing volume perturbation of the training data, where each recording in the training data was scaled with a random variable drawn from a uniform distribution over $[\frac{1}{64}, 8]$, emulates

[2] The notation $\{-7,2\}$ means we splice together the input at the current frame minus 7 and the current frame plus 2.

[3] We would like to thank Mitsubishi Electric Research Laboratories (MERL), for providing the RWCP database.

mean shifts in the MFCC domain. The volume perturbation was done on the reverberated speech audio. Results pertaining to volume perturbation are tagged as such in Section 4.

2.4. iVector Extraction

In this section we describe the iVector estimation process adopted during training and decoding. We discuss issues in estimating iVectors from noisy unsegmented speech recordings, and in using these noisy estimates of iVectors as input to neural networks.

We noticed that the iVector adaptation was not sufficiently effective in adapting to test signals that had substantially different energy levels than the training data. For the results reported here, this issue was resolved by normalizing the test-signal energies to be the same as the average of the training data. We compare this approach with volume perturbation approach (2.3.1) in Section 4.

2.4.1. iVector Extraction during training

The iVector estimator was trained on a 100 hour subset of training data: this includes the training of the Gaussian mixture model used for the UBM, and the estimation of the total-variability (T) matrix. Then, for the entire training data, iVectors were estimated. In order to ensure sufficient variety of the iVectors in the training data, rather than estimating a separate iVector per speaker we estimate them in an online fashion, where we only use frames prior to the current frame (for some arbitrary ordering of the utterances). We reset this history every two utterances, so that we still have some training-data variety even when there are only a few speakers.

2.4.2. iVector extraction during decoding

During decoding, the constraints of online extraction were not enforced and iVectors were estimated in an offline fashion from statistics accumulated over fairly large portions of the speaker's data (at least 60 seconds).

The prior term in the iVector extraction is quite important when applying these iVector based methods to data that is dissimilar to the training data. In our iVector estimation we always scale the per-frame posteriors by 0.1 (equivalent to scaling the prior term up by 10). For the ASpIRE challenge we made a further modification: if the total count of (scaled) statistics for iVector extraction exceeds a predefined limit (75 for these experiments), we scale the statistics down to that value, which again is equivalent to scaling the prior term up. Due to the posterior scale of 0.1, this effect kicks in after we exceed 750 frames of features.

2.4.3. iVectors from reliable speech segments

In the current LVCSR task (see Section 4), audio recordings 5-10 minutes in length were provided without speech end-point information. The recordings had long duration of contiguous silence, similar to single channel recordings of conversational telephone speech. We found empirically that excluding the silence from the statistics for iVector estimation was very helpful. Even keeping a small amount of silence around every speech segment (similar to the amount we saw in training) was harmful; possibly the nature of the silence in the ASpIRE test data was so different from what was seen in the artificially reverberated and noise-added training data, that it affected the iVector in unexpected ways. Hence only feature vectors from the speech segments were used for iVector estimation. We explored two techniques to detect speech segments, which are described below.

1. iVectors using two-pass decoding

In the first method, we perform a first-pass decode of the audio data using iVectors derived from both speech and non-speech regions. Reliable speech segments are identified from this first-pass decode. Audio segments corresponding to words with confidence measures of 1.0 (derived from lattice posteriors) and with durations less than one second were considered reliable (over half the words recognized had a confidence of at least 1.0). We also excluded the words "mm" and "mhm". A second pass decode was then performed using the iVectors estimated from these reliable speech segments. This led to 8.9% relative improvement in WER, versus using all the data for iVector estimation.

2. iVectors using Voice Activity Detection (VAD)

As two-pass decoding is computationally expensive, we attempted a GMM-based Voice Activity Detection (VAD) to detect regions of speech. The VAD method used is a hybrid feature and model-based method inspired by [19]. It works by training a HMM-GMM system with 3 GMMs - *Silence*, *Speech* and *Noise* on approximately 10 minute long chunks of the audio recordings. The features used with the GMMs are 12 (excluding C0) mean-normalized MFCCs along with their deltas and delta-deltas and zero-crossing rates along with its delta and delta-delta. The GMMs are initially bootstrapped using frame-alignments of the augmented training set described in the previous section. For this, the phones are mapped into 3 classes – silence, speech and noise. The GMMs are iteratively trained using Viterbi decoding followed by re-estimation.

For the first few iterations of training, only the low-energy and high zero-crossing rate frames from the non-speech frames are selected for *Silence* GMM and *Noise* GMM training respectively. The later iterations use all the frames of the respective classes. We have used the same training procedure as in [19]. The number of Gaussians in each model is increased every iteration until the number of Gaussians for *Silence*, *Noise* and *Speech* are 7, 18 and 16 respectively.

A Bayesian information criterion (BIC) is used to determine if the *Noise* GMM is to be retained. If the *Noise* GMM is to be removed, then the entire process is repeated using only the *Silence* and *Speech* GMMs bootstrapped from the augmented training data.

The regions selected as speech by the VAD are used for iVector estimation and a single-pass decode is performed using these iVectors. With this method, we were able to improve the speed over the two-pass decoding system by a factor of $\sim$ 2, while keeping the WER degradation on the dev set to 1.2%, relative.

2.5. Training

The paper follows the training recipe detailed in [14]. It uses greedy layer-wise supervised training, preconditioned stochastic gradient descent (SGD) updates, an exponentially decreasing learning rate schedule and *mixing-up*. Parallel training of the DNNs using up to 18 GPUs was done using the model averaging technique in [10].

Sequence training was done on the DNN, based on a state-level variant of the Minimum Phone Error (MPE) criterion, called sMBR [20] . The training recipe mostly follows [21], although it has been modified for the parallel-training method.

In the sMBR objective function insertion errors are not penalized, which could lead to larger number of insertions when decoding with sMBR trained acoustic models. Correcting this asymmetry in the sMBR objective function, by penalizing insertions, was shown to improve performance of sMBR models by 10% WER, relative [13]. This modified objective function was used in this paper.

To compute the context-dependent state pseudo-likelihoods from the posteriors estimated by the neural network, the posteriors are divided by a prior. We found that the method of using the mean posterior (computed over a subset of the training data) as the prior [22] gave an improved performance when decoding with sMBR-trained models, so we used this method.

A speaker-adapted GMM-HMM acoustic model similar to the one described in [21] was used to generate context-dependent state alignments for training neural networks. Alignments for clean speech were generated using the GMM-HMM system, using clean data as recommended in [23, 24].

3. LANGUAGE MODEL

3.1. Lexicon

CMUdict (0.7a) was used as training lexicon in our experiments, and the vocabulary was restricted to the words that appear in the training transcripts. CMUdict comes with multiple pronunciations for some words, therefore we estimate the pronunciation probabilities during the training. We also model inter-word silence probabilities as described in [25]. The statistics for modeling pronunciation and silence probabilities were estimated from training data alignment, and they were later encoded into the lexicon finite state transducer during the decoding.

We estimate pronunciation probabilities for a word with multiple pronunciations via simple relative frequency, with proper smoothing techniques [26, 27, 28]. Directly using the simple relative frequency however has an undesirable consequence that a word with several equiprobable pronunciations is unfairly handicapped w.r.t words that have a single pronunciation: e.g. the past tense of "read" w.r.t the color read "red". Max-normalization, whereby the pronunciation probabilities are scaled so that the most likely pronunciation of each word has "probability" 1, has been found helpful in speech recognition [29] We therefore applied max-normalization for pronunciation probabilities in our work.

For a given sequence of words, we assume there is either a silence or non-silence event between two consecutive words. Since such an event usually depends on the neighbouring words, we further assume that it only depends on the two surrounding words, i.e., we model the event using $P(s \mid w.p_i, w'.p_j)$ and $P(n \mid w.p_i, w'.p_j)$, where $w.p_i$ and $w'.p_j$ are the surrounding pronunciations, s and n represent silence and non-silence event. For computation simplicity, we decompose this into two parts: (i) probability of inter-word silence (or non-silence) following the pronunciation, and (ii) probability of inter-word silence (or non-silence) preceding the pronunciation. Details of how we compute those probabilities can be found in [25].

3.2. N-gram LM

N-gram language models were used in the decoding to generate word lattices. A trigram language model (LM) was first trained on the 3M words of the training transcripts, which was later interpolated with another trigram LM trained on 22M words of the Fisher English transcripts (LDC2004T19 and LDC2005T19). The same process is repeated for building a 4gram LM. We used SRI's language modeling toolkit SRILM [30] for building our LMs, with Kneser-Ney smoothing. The final trigram LM has 1.6M trigrams and the 4gram LM has 1.7M 4grams.

We directly used the trigram LM for decoding. The 4gram LM, however, was only used for rescoring the lattices generated by the trigram LM. We did not use the 4gram LM for lattice generation for reasons of memory efficiency. We implemented an efficient in-memory representation of the ngram LM, with which we can build grammar FST on the fly when rescoring, the same technique as described in [31].

3.3. RNN-LM rescoring

3.3.1. training

We used the RNNLM toolkit (0.3e) [32] to train our recurrent neural network language models (RNN-LMs). We first selected the most frequent 40k words from training transcripts as the language model vocabulary. We then took 10k utterances from the training transcripts as a heldout set, and used the rest training transcripts for RNN-LM training. We used 200 hidden units for the neural network, and set the number of word classes to 350. The 40k vocabulary words were assigned to the 350 word classes according to their unigram frequency in the training corpus. The maximum order of the n-gram features [33] was set to 4, and truncated back-propagation through time (BPTT) [34] was performed during training with a step size of 2.

Words from the training transcripts that were not selected in the RNN-LM vocabulary were mapped to a special word "<RNN_UNK>" before training. The unigram probabilities of those words were also collected from training transcripts which later served as penalties when we compute the likelihood of sentences containing those words. For words that were not in the training transcripts nor in the RNN-LM vocabulary, a fixed unigram probability of $1e-7$ was used instead.

3.3.2. N-best rescoring

We followed the conventional N-best RNN-LM rescoring procedure. N-best hypotheses were first extracted from lattices, with their associated acoustic score, original language model score, graph cost as well as frame level alignments. The RNN-LM likelihood was then computed for each hypothesis, with out-of-vocabulary words properly penalized as described in Section 3.3.1. In our experiments, we found interpolating the RNN-LM likelihood with the original language model did not help much in our particular setup, so we simply replaced the original language model score with the RNN-LM score. The acoustic score, RNN-LM score, graph cost and the frame level alignment of all the hypotheses were then packed back to create a new lattice, with which we ran the decoding. It worth mentioning that it is important to generate the N-best hypotheses with the optimal acoustic scale.

3.3.3. Lattice rescoring

One drawback of applying RNN-LMs on N-best list is the N-best list only covers a subset of the hypotheses from the original lattice. Therefore if the original language model is not powerful enough, and the correct word falls out of the N-best list, there is no way for RNN-LM to recover it. A simple solution is to generate as many hypotheses as possible, which of course will increase the decoding cost.

Applying RNN-LM rescoring directly on the lattice however can increase the lattice size exponentially since the number of distinct RNNLM context states will grow exponentially. A generally solution is to derive appropriate equivalence classes for context states. In [35], two different methods to cluster the context states were evaluated: (i) clustering context states using n-gram history, and (ii) clustering context states based on the context vector distance. The authors were able to get the same performance from these two methods. In this work, we cluster the context states using n-gram history since this is computationally cheap.

We implemented our lattice based RNN-LM rescoring within the weighted finite state transducer (WFST) framework. During the rescoring, a grammar WFST is generated on-the-fly, whose states each correspond to a unique n-gram sequence. The weight of the arc given the states is given by $P(w_n|w_1, \ldots, w_{n-1})$, where w_n is the word on the arc, and $w_1, \ldots, w_{n-1}$ is the n-gram history that the state corresponds to, which can be computed from RNN-LM. In the actual implementation we store the RNN-LM context vector instead the word sequence, so that the RNN-LM can compute $P(w_n|h)$ directly, where h is the context vector of the word sequence whose latest $n-1$ words are $w_1, \ldots, w_{n-1}$. Ideally, we would like h to correspond to the best possible sequence in the lattice entering the state. Our preliminary results however shows that we may not benefit a lot from using the context vector from the best possible word sequence entering the state. Therefore in our current implementation we simply use the context vector from the first word sequence we see whose latest $n-1$ words correspond to the state.

4. RESULTS

Two data sets *dev* of 5 hrs and *dev-test* of 10 hrs were provided as part of ASpIRE challenge. Each set is composed of 10 minute recordings. The end points for the speech portions of the recording were also provided for the *dev* set. However in order to emulate the decoding scenario of *dev-test*, we report performance on *dev* set without the knowledge of segment information.

Decoding the entire 10 minute recording as one segment is not possible due to round-off error in the decoder. We segmented the recordings into 10 seconds long chunks, shifted by 5 seconds each time. There was no attempt to make the chunk boundaries coincide with silence. We reasoned that if a recording is cut in the middle, only the part of the transcript near the cut point will be affected, so we filtered the transcripts by removing words whose midpoints were within 2.5 seconds of the edge of its chunk of origin, before combining them into a single long transcript. We tuned the optimal acoustic scale on *dev* set, which typically falls between 0.08 and 0.1.

4.1. Neural network architecture and data augmentation

The TDNN had 6 layers, of which 3 layers (not counting the input layer) were subject to frame splicing across multiple time offsets. Three different systems TDNN-A, TDNN-B and TDNN-C corresponding to the three different input contexts of $[t-13, t+9]$ frames, $[t-16, t+12]$ frames and $[t-22, t+12]$ were used in the comparison. The splicing configuration of the TDNN-A system was $[-2, 2]$, $\{-1, 2\}$, $\{0\}$, $\{-3, 3\}$, $\{-7, 2\}$, $\{0\}$ (where the $\{0\}$ layers are conventional, non-splicing hidden layers). The TDNN-B and TDNN-C systems were as TDNN-A except replacing $\{-7, 2\}$ with $\{-10, -7, 2, 5\}$ and $\{-16, -7, 2, 5\}$ respectively. In all hidden layers the p-norm input and output dimensions were 4000 and 400 respectively.

Results corresponding to this comparison are presented in Table 1. Comparing the TDNN-A systems trained on clean and reverberant data it can be seen that multi-condition training data is critical. Comparing TDNNs A, B and C with different input contexts it can be seen that input context of $[t-16, t+12]$, was optimal for training on reverberant speech. This result can be compared with the input context of $[t-13, t+9]$ found optimal for recognition of non-reverberant speech in [3]. This additional context could be necessary when training on reverberant data to compensate for the late reverberations. The use of even larger temporal contexts (TDNN-C) did not lead to better results. However it was interesting to note that use of larger contexts was not detrimental to the same extent as seen in other speech recognition tasks with non-reverberant data [3]. Further a 6-layer DNN with input context of $[-16, 12]$, and with the same p-norm input/output dimensions was trained on reverberant data. Using the DNN architecture, in place of TDNN, led a 7% relative increase in WER.

Table 1: Comparison of input contexts and training data augmentation, used for training the TDNNs

Acoustic Model	context	training data	*dev* WER
TDNN A	[-13, 9]	clean	47.6
TDNN A	[-13, 9]	rvb	31.7
TDNN B	[-16, 12]	rvb	30.8
TDNN B	[-16, 12]	rvb + sp	31.0
TDNN C	[-22, 12]	rvb + sp	31.1
DNN	[-16, 12]	rvb	33.1

rvb : reverberation of training data using real world RIRs
sp : speed perturbation of data prior to reverberation

These systems were trained on data generated from two different types of data augmentation techniques which are reverberation (rvb) and speed perturbation (sp). Speed perturbation which was shown to be advantageous across several LVCSR tasks [18], was not helpful in the current task. Training for more epochs improved the performance of the TDNN-B (rvb+sp) system; however, it just matched the TDNN-B (rvb) system.

Table 2 shows the impact of volume normalization of test data on system performance. It can be seen that even with volume perturbed training data the acoustic model was not able to tackle the volume mismatch observed in ASpIRE test data. However volume perturbation training led to a relative improvement of 13% when dealing with non-normalized test data. Volume normalization of the test data led to a relative improvement of 19.5% in WER when using the TDNN-B system trained on non-volume perturbed data.

Table 2: Comparison of systems w/ & w/o volume perturbed training data and w/ & w/o volume normalized test data

Acoustic Model	Training Data	Test Data	*dev* WER
TDNN B	rvb		38.3
TDNN B	rvb	vol. norm.	30.8
TDNN B	rvb +vp		33.3
TDNN B	rvb +vp	vol. norm.	30.9

rvb : reverberation of training data using real world RIRs
vp : volume perturbation of data after reverberation

Figure 2 shows the reduction in WER for the TDNN-B (rvb) system during different stages of training. It can be seen that even with 100 iterations[4], which corresponds to $\sim$ 8 hours of training time (wall-clock time) the WER error is 33.4%. (See [10] for greater detail on distributed optimization technique used here).

4.2. iVectors

Table 3 compares systems trained with and without iVectors, and different types of iVector extraction methods. We tried four different ways to extract iVectors: (i) extracting iVectors from both speech and non-speech frames, (ii) extracting iVectors from speech frames, but in a online mode (i.e., for the current frame, only use speech

[4] each iteration corresponds to 40,000 training examples per distributed optimization instance

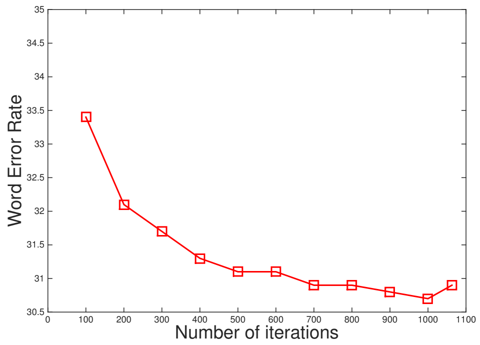

Fig. 2: *dev* WER as training progresses

Table 3: Comparison of systems with and without iVectors

Acoustic Model	*dev* WER
TDNN B w/o iVectors	34.8
TDNN B + iVectors[1]	33.8
TDNN B + iVectors[2]	33.1
TDNN B + iVectors[3]	30.8
TDNN B + iVectors[4]	31.2

[1] estimated on speech and non-speech frames
[2] estimated on speech frames online
[3] estimated on speech frames offline
[4] estimated on speech frames from VAD

frames before the current frame) (iii) extracting iVectors from speech frames, but in a offline mode (a first pass decoding was used to identify the speech regions) and (iv) extracting iVectors from speech frames computed from VAD. From the table it's clear that it is beneficial to use iVectors in our system, and it is also critical to extract iVectors only from speech frames.

4.3. Pronunciation and Silence Probabilities

Table 4: Impact of pronunciation and silence probabilities

Model	*dev* WER
TDNN B*	32.1
TDNN B* + pronprob	31.6
TDNN B* + pronprob + silprob	30.8

* without pronunciation and silence probabilities.

Table 4[5] shows performance of using pronunciation and inter-word silence probabilities in the lexicon FST during decoding. As it's shown in the table, it is generally helpful to model pronunciation and silence probabilities in the lexicon FST.

4.4. RNN-LMs

Our RNN-LM rescoring results are shown in Table 5. We do the rescoring on both N-best lists and lattices. For N-best list rescoring, we tried to keep 100, 500 and 1000 best paths respectively, and we found it was enough to keep 500 best paths. For lattice rescoring, we used the RNN-LM to compute likelihood over a fixed number of context and it's shown in the table that keeping context of 5gram is sufficient in this particular case.

Table 5: Impact of RNN-LM rescoring on TDNN B model

Model	*dev* WER
4gram LM Baseline	30.8
RNN-LM N-best top 100	30.2
RNN-LM N-best top 500	29.9
RNN-LM N-best top 1000	29.9
RNN-LM lattice max 4gram	29.9
RNN-LM lattice max 5gram	29.8
RNN-LM lattice max 6gram	29.8

4.5. Sequence Training

Table 6: Results with sequence training of TDNN models

Acoustic Model	*dev* WER
TDNN A	31.7
TDNN A + sequence training[1]	34.0
TDNN A + sequence training[2]	30.6
TDNN B	30.8
TDNN B + sequence training[2]	29.5
TDNN B + sequence training[2,3]	29.1

[1] with sMBR criterion
[2] with modified sMBR criterion
[3] prior-adjustment

Table 6 shows results of TDNNs using sequence training. The standard sMBR criterion was detrimental to the performance; but using the modified sMBR criterion, gains were observed on *dev* set. However these did not translate to *dev-test* set. With sequence training there was 4.2% relative improvement on *dev* set and 4.3% relative decrease on *dev-test* set. Further it can be seen that using priors computed from mean posteriors led to an improvement in the performance. TDNN-B with modified sequence training and modified prior computation is used in the next section.

4.6. Comparison across test-sets

From Table 7 it can be seen that sMBR training has mixed results. On careful analysis of the results on evaluation set it was observed that the sMBR system outperformed cross-entropy system for 70% of the 120 speakers. However the WER drastically increased for the other 30%. It was also observed that the sMBR system was prone to insertion errors, despite the use of the modified-sMBR objective function.

Table 7: Comparison across test-sets

Model	*dev*	*test*	*eval*
TDNN B*	30.8	27.7	44.3
TDNN B + RNN-LM	29.8	**26.5**	**43.4**
TDNN B + sMBR	29.1	28.9	43.9
TDNN B + sMBR + RNN-LM	**28.3**	28.2	**43.4**

* submitted to the evaluation

5. CONCLUSIONS

Using a combination of simulated multi-condition training data, TDNN acoustic models with wide temporal contexts, iVector adaptation and RNN-LMs we were able to build a state-of-the-art system which was able to tackle mismatch conditions in far-field speech recognition task.

[5] All the other results shown in this paper are with pronunciation and silence probabilities

6. REFERENCES

[1] T. Yoshioka, A. Sehr, M. Delcroix, K. Kinoshita, R. Maas, T. Nakatani, and W. Kellermann, “Making machines understand us in reverberant rooms: Robustness against reverberation for automatic speech recognition,” *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 114–126, Nov 2012.

[2] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, “Phoneme recognition using time-delay neural networks,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 3, pp. 328–339, Mar. 1989.

[3] 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.

[4] S. Xue, O. Abdel-Hamid, H. Jiang, L. Dai, and Q.-F. Liu, “Fast Adaptation of Deep Neural Network based on Discriminant Codes for Speech Recognition,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. PP, no. 99, pp. 1–1, 2014.

[5] M. Karafiat, L. Burget, P. Matejka, O. Glembek, and J. Cernocky, “iVector-based discriminative adaptation for automatic speech recognition,” in *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*. IEEE, Dec. 2011, pp. 152–157.

[6] 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.

[7] M. J. Alam, V. Gupta, P. Kenny, and P. Dumouchel, “Use of multiple front-ends and i-vector-based speaker adaptation for robust speech recognition,” *Proc. of IEEE REVERB Workshop*, 2014.

[8] *Automatic Speech recognition In Reverberant Environments (ASpIRE) Challenge*, 2015 (accessed March 23 , 2015), http://www.iarpa.gov/index.php/working-with-iarpa/prize-challenges/306-automatic-speech-in-reverberant-environments-aspire-challenge.

[9] C. Cieri, D. Miller, and K. Walker, “The fisher corpus: a resource for the next generations of speech-to-text.” in *LREC*, vol. 4, 2004, pp. 69–71.

[10] D. Povey, X. Zhang, and S. Khudanpur, “Parallel training of Deep Neural Networks with Natural Gradient and Parameter Averaging,” in *Proceedings of the ICLR Workshop*, 2015.

[11] S. B. Davis and P. Mermelstein, “Comparison of parametric representation for monosyllabic word recognition in continuously spoken sentences,” *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 28, no. 4, pp. 357–366, 1980.

[12] N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, “Front-end factor analysis for speaker verification,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788 –798, May 2011.

[13] V. Peddinti, G. Chen, D. Povey, and S. Khudanpur, “Reverberation robust acoustic modeling using i-vectors with time delay neural networks,” in *Proceedings of Interspeech*, 2015.

[14] X. Zhang, J. Trmal, D. Povey, and S. Khudanpur, “Improving deep neural network acoustic models using generalized maxout networks,” in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, May 2014, pp. 215–219.

[15] S. Nakamura, K. Hiyane, F. Asano, T. Nishiura, and T. Yamada, “Acoustical sound database in real environments for sound scene understanding and hands-free speech recognition.” in *LREC*, 2000.

[16] K. Kinoshita, M. Delcroix, T. Yoshioka, T. Nakatani, A. Sehr, W. Kellermann, and R. Maas, “The reverb challenge: A common evaluation framework for dereverberation and recognition of reverberant speech,” in *Applications of Signal Processing to Audio and Acoustics (WASPAA), 2013 IEEE Workshop on*. IEEE, 2013, pp. 1–4.

[17] M. Jeub, M. Schafer, and P. Vary, “A binaural room impulse response database for the evaluation of dereverberation algorithms,” in *Digital Signal Processing, 2009 16th International Conference on*. IEEE, 2009, pp. 1–5.

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

[19] M. Huijbregts and F. de Jong, “Robust speech/non-speech classification in heterogeneous multimedia content,” *Speech Communication*, vol. 53, no. 2, pp. 143 – 153, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167639310001421

[20] M. Gibson, “Minimum bayes risk acoustic model estimation and adaptation,” Ph.D. dissertation, University of Sheffield, 2008.

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

[22] V. Manohar, D. Povey, and S. Khudanpur, “Semi-supervised maximum mutual information training of deep neural network acoustic models,” in *Proceedings of INTERSPEECH*, 2015.

[23] M. Delcroix, T. Yoshioka, A. Ogawa, Y. Kubo, M. Fujimoto, N. Ito, K. Kinoshita, M. Espi, T. Hori, T. Nakatani *et al.*, “Linear prediction-based dereverberation with advanced speech enhancement and recognition technologies for the reverb challenge,” in *Proc. IEEE REVERB Workshop*, 2014.

[24] Y. Tachioka, T. Narita, F. Weninger, and S. Watanabe, “Dual system combination approach for various reverberant environments with dereverberation techniques,” in *Proc. of IEEE REVERB Workshop*, 2014.

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

[26] T. Hain, “Implicit modelling of pronunciation variation in automatic speech recognition,” *Speech Communication*, vol. 46, no. 2, pp. 171–188, 2005.

[27] B. Peskin, M. Newman, D. McAllaster, V. Nagesha, H. Richards, S. Wegmann, M. Hunt, and L. Gillick, “Improvements in recognition of conversational telephone speech,” in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 1. IEEE, 1999, pp. 53–56.

[28] T. Hain, P. Woodland, G. Evermann, and D. Povey, “The CU-HTK march 2000 Hub5e transcription system,” in *Proceedings of Speech Transcription Workshop*, vol. 1, 2000.

[29] E. Fosler, M. Weintraub, S. Wegmann, Y.-H. Kao, S. Khudanpur, C. Galles, and M. Saraclar, "Automatic learning of word pronunciation from data," in *Proceedings of the International Conference on Spoken Language Processing*, 1996.

[30] A. Stolcke *et al.*, "SRILM - an extensible language modeling toolkit." in *Proceedings of INTERSPEECH*, 2002.

[31] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an ASR corpus based on public domain audio books," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015.

[32] T. Mikolov, S. Kombrink, A. Deoras, L. Burget, and J. Cernocky, "Rnnlm-recurrent neural network language modeling toolkit," in *Proc. of the 2011 ASRU Workshop*, 2011, pp. 196–201.

[33] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černockỳ, "Strategies for training large scale neural network language models," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. IEEE, 2011, pp. 196–201.

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," DTIC Document, Tech. Rep., 1985.

[35] X. Liu, Y. Wang, X. Chen, M. J. Gales, and P. C. Woodland, "Efficient lattice rescoring using recurrent neural network language models," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 4908–4912.