DEEP BI-DIRECTIONAL RECURRENT NETWORKS OVER SPECTRAL WINDOWS

Abdel-rahman Mohamed, Frank Seide, Dong Yu, Jasha Droppo, Andreas Stolcke, Geoffrey Zweig, Gerald Penn[*]

Microsoft Research, [*]University of Toronto

ABSTRACT

Long short-term memory (LSTM) acoustic models have recently achieved state-of-the-art results on speech recognition tasks. As a type of recurrent neural network, LSTMs potentially have the ability to model long-span phenomena relating the spectral input to linguistic units. However, it has not been clear whether their observed performance is actually due to this capability, or instead if it is due to a better modeling of short term dynamics through the recurrence. In this paper. we answer this question by applying a windowed (truncated) LSTM to conversational speech transcription, and find that a limited context is adequate, and that it is not necessaary to scan the entire utterance.

The sliding window approach allows not only incremental (online) recognition with a bidirectional model, but also frame-wise randomization (as opposed to utterance randomization), which results in faster convergence.

On the SWBD/Fisher corpus, applying bidirectional LSTM RNNs to spectral windows of about 0.5s improves WER on the Hub5'00 benchmark set by 16% relative compared to our best sequence-trained DNN. On an extended 3850h training set that that also includes lectures, the relative gain becomes 28% (Hub5'00 WER 9.2%). In-house conversational data improves by 12 to 17% relative.

Index Terms— Recurrent networks, LSTM, Deep learning, acoustic modeling

1. INTRODUCTION

Neural network techniques for acoustic modeling can be roughly divided into two categories: feed-forward networks which operate on a fixed window of frames, and recurrent networks which recursively process the entire utterance. Deep feed-forward neural networks (DNNs) with context dependent triphone-state targets have been tremendously successful, improving the word-error rate over comparable GMM baselines by up to 40% relative on the Switchboard conversational transcription benchmark [1, 2, 3, 4]. Recurrent Neural Network (RNN) acoustic models [5, 6, 7] have further improved the results, as has speaker normalization and noise compensation [8, 4].

DNNs model spectral windows using an exponential number of distinct states that share latent factors [9, 4]. This allows DNNs to generalize to unseen events better than Gaussian Mixture Model (GMM) acoustic models which do not have the same sharing across the models for different triphone states. DNN models can be further improved by applying convolution and pooling over time and frequency as in Convolutional Neural Networks (CNNs) [10, 11]. The convolution opertation allows the networks greater flexibility in modeling small temporal and spectral shifts in speech data resulting from different speakers and speaking speeds as well as band-limited noise, while enjoying the same distributed representation power of DNNs. In both DNN and CNN acoustic models, limited spectral windows under 0.5s are typically used as inputs.

In contrast to feed-forward networks, Recurrent Neural Network (RNN) acoustic models [5, 6] scan the entire utterance, and the resulting probability distribution over HMM-states for each input acoustic frame are conditioned on all frames observed prior to the current time t. With their built-in memory cells, Long Short-Term Memory (LSTM) RNNs [6] are capable of relating distant past to current acoustic events, achieving drastically better performance on different acoustic modeling tasks [6, 12, 7] compared to baseline DNNs and RNNs.

In light of the exceptional performance of RNNs, it is interesting to try to understand the reasons for this performance. There are at least two reasonable explpanations: (1) RNNs sequentially process the entire input utterance conditioning each input frame on the entire history up to the current point in time. Learning the global structure of input sequences could allow the network to learn a kind of phonetic sequence model, i.e. an implicit probability distribution of each sound conditioned on previously observed sounds. Also, the network could be marginalizing over other global information in the sequence that is unnecessary for the recognition task, like session and speaker characteristics. (2) Rather than explaining the entire spectral window at once by the hidden layer representation, as it is the case for DNNs, RNNs consume only one frame at a time using input-to-hidden connections while learning the relationship of the current frame to its previous (and future, if bi-directional [13]) frames via lateral hidden-to-hidden connections. Dividing the task into two sets of weights, input-to-hidden and hidden-to-hidden, allows different hidden units to "conspire" to jointly map complicated input speech patterns into simpler abstractions that generalize better to unseen events.

Under the second view, RNNs act as an extreme case of CNNs with the convolution operation applied along the time axis using single frame inputs, while the pooling operation is replaced by a learned hidden-to-hidden matrix that automatically adjusts the pooling window as needed. (For convolution over the frequency axis, we can imagine an RNN looping over the frequency axis where the learned hidden-to-hidden weights act as a generalization of the heterogeneous pooling mechanism [14].)

This paper examines the second hypothesis by limiting the amount of input that the RNN is allowed to see, exactly as is done with DNN and CNN inputs. The empirical results lead us to conclude that RNNs owe their good performance to their recurrent way of processing input frames, rather than to having access to the entire utterance history. The important discriminative information is available in the local neighborhood of each input frame, but DNN and CNN acoustic models fail to match RNNs in extracting such information.

Along the same lines, the authors in [15] proposed to apply a simple RNN to speaker-adapted features followed by multiple layers of feedforward neural networks. The new composite model showed

978-1-4799-7291-3/15/$31.00 ©2015 IEEE

about 5% relative WER improvement over the DNN only baseline on a standard ASR task.

In the next section, we will describe the specific windowed RNN used in this paper. We will then discuss the practical implementation, including parallelization of training with 1-bit SGD (section 3) and speeding up decoding and enabling sequence training by frame grouping and "jitter training" (section 4), followed by experimental results.

2. RECURRENCE OVER SPECTRAL WINDOWS

The main aim of this paper is to show that important discriminant information has been readily available to the DNN and CNN acoustic models, but they failed to extract it. Whereas RNNs succeeded due to the way they process input frames recurrently one-by-one, and by simplifying the task into a frame-embedding task using the input-to-hidden weights followed by a recurrence step that relates individual embedding to neighboring ones using the hidden-to-hidden weights.

2.1. Model Structure

Like DNNs and CNNs, the windowed RNN in this paper predicts the probability of a speech state (such as a tied triphone state, senone) in a speech frame t by processing an input spectral window consisting of speech frames $t - T$ to $t + T$.

Specifically, we use a *deep bi-directional LSTM RNN* [6] that recurrently processes all input frames τ inside the window to predict the HMM states of the desired center frame t. The model consists of several stacked bi-directional LSTM RNNs (up to 6 in our case), where each bi-directional layer has two different hidden-to-hidden weight matrices, one for each direction, i.e. left-to-right and right-to-left.

Each layer's hidden states of all frames in the window, of both directions, are fed as inputs to the respective frames of the next layer. Except for the input layer, each hidden layer therefore has four different input-to-hidden weight matrices to process inputs from the two input streams to the current layer forward and backward streams.

At the top sits a softmax layer that takes the top hidden layer's forward and backward hidden state of the center frame as its input. There is no fully connected hidden layer. It's RNNs all the way down.

The LSTM units [16] are of today's standard variant, with input, forget, and output gates controlling the flow from and to the cell that stores the state of each unit (e.g., [6]).

2.2. Uni- vs. bi-directional RNNs

Why bi-directional? Although speech is not symmetric, coarticulation happens in both directions. For a uni-directional RNN to capture any right context, the network would have to propagate the information from the predicted frame t itself—arguably the most pertinent information—with high fidelity through multiple steps of recurrence. Although LSTMs offer substantially better memorization of acoustic events in distant past than simple RNNs, LSTMs can still potentially attenuate the stored signal in the cell. For uni-directional networks, acoustic events occurring by the beginning of spectral window may get too diluted to affect the decision, and the target frame might not be predicted with high temporal precision. Bi-directional RNNs offer a more stable alternative.

This was experimentally confirmed in preliminary tests on a 400h-subset of our main training set described in the results section, using phoneme-loop decoding. (Both systems also used mixture of softmaxes, see next section.) A bi-directional model with a hidden dimension of 512 outperformed a uni-directional model with twice the hidden dimension by 0.6% absolute phoneme error rate (4% relative).

2.3. Mixture of Softmaxes

We also experimented with a refinement aimed at squeezing the most out of the input window: a mixture-of-softmax structure that acts on the bi-directional hidden representation extracted at *every* time step τ in the $2T + 1$ frames long window.

Two different hidden-to-output weight matrices, which are shared across all time steps τ, process hidden representations from both forward and backward streams leading to a matrix O of $(2T + 1)$ columns of unnormalized log output activations for the K HMM states. These are combined through a mixing weight matrix M of size $(2T + 1) \cdot K$ to produce the final prediction vector P:

$$P_k \quad = \quad \frac{\exp(\sum_{\tau=t-T}^{t+T} m_{\tau k} \cdot o_{\tau k})}{\sum_{j=1}^{K} \exp(\sum_{\tau=t-T}^{t+T} m_{\tau j} \cdot o_{\tau j})} \qquad (1)$$

M is trained jointly with the other network weights. The mixture softmax is designed to force different output HMM states to attend differently to different positions within the input window. It also allows for using large input spectral windows because it ensures that every input frame would have a say on predicting the target label regardless of its relative distance of the label location in the window.

In preliminary tests, the mixture of softmaxes was effective: It had a 4% relative better word-error rate for a 2-hidden-layer network trained on 400h. Unfortunately, because it has very large memory requirements, we were not able to use it in the large-scale experiments in this paper. Instead, as stated above, we train the network to predict HMM states from the hidden representations at the center frame only. We hope that future developments will allow us to use this method effectively also for large-scale production model training.

3. TRAINING WINDOWED RNNS

The model are trained from random initialization using back-propagation. We use AdaGrad and a learning-rate profile that was originally optimized for DNN training, but turned out to work well for RNNs as well. Two aspects of our implementation, though, warrant additional discussion.

3.1. Subsampling

Training on large volumes of data that does not fit a server's RAM requires to process data in chunks. Our training software uses a large rolling window over randomized blocks of about 15 minutes of data. It then randomly samples from the rolling window to create training minibatches. Sampling is done such that each frame is used exactly once as a center frame per data pass.

This also means that every frame is used exactly once in every *non*-center position of the context window. But because, unlike DNNs, RNNs apply the same model parameters to all frames in the window, this causes overfitting to the loaded chunk in RAM. Our remedy is to sub-sample the data further to only a small fraction (5%) of loaded data. This leads to a very notable WER difference.

Table 1. *Scalability of 1-bit SGD (6 hidden layers 20+8+20 jitter): processing speed for a minibatch size of 8192.*

nodes x GPUs	1x1	1x4	2x4	1x8	2x8	4x8	8x8
speed [ksps]	0.13	0.5	0.9	0.9	1.7	2.8	4.6
efficiency	100%	96%	87%	87%	82%	67%	55%

3.2. Parallel Training Using 1-Bit SGD

The windowed RNNs' benefit of rapid convergence comes at a cost: very large computational cost per sample. Compared to a feed-forward DNN, a bi-directional LSTM with about the same number of parameters over a window of 41 frames requires 41 times as many multiply-and-accumulate operations, because every model parameter is applied repeatedly to *every frame* in the window. For our best model configuration, processing 24h of training data costs on the order of 18 hours.

As a consequence, training of windowed RNNs is only feasible if a method of parallelization is used.

We apply our previously proposed 1-bit SGD parallelization technique [17]. It implements data-parallel training on the minibatch level: Minibatches are distributed over multiple computation nodes, and for each minibatch the sub-gradients are aggregated across all nodes (an "all-reduce" operation).

Unfortunately, a direct implementation of this approach cannot lead to meaningful speed-ups, because the time it takes to exchange the gradients is significantly larger than the time it takes to compute a sub-batch on a compute node.

Our method alleviates this by reducing bandwidth in two ways: *automatic minibatch-size scaling* and *1-bit quantization of gradients*.

Automatic minibatch-size scaling means that at regular intervals (72h of training data), the trainer automatically adjusts the minibatch size to the largest possible. Increasing minibatch size by a certain factor means reducing the number of data exchanges and thus bandwidth by that same factor [18]. The algorithm tries out larger minibatch sizes on a small subset of upcoming data (45 minutes), where learning rate and momentum parameters are normalized such that the contribution from each sample remains the same (specifically, we do not divide the gradient by the number of samples, and define momentum as a unit-gain low-pass filter with a fixed time constant). We find that there is a maximum minibatch size up to which convergence does not change, but above which convergence slows or breaks down. The process aims to select that maximum feasible minibatch size, as to minimize bandwidth.

Secondly, the all-reduce step exchanges gradients rather than model parameters because gradients are amenable to data compression. In [17], we showed that gradient values can be aggressively quantized to but a single bit, if one carries over the quantization error from one minibatch to the next. Specifically, each time we quantize a sub-gradient, we compute and remember the quantization *error*, and then add it to the *next* minibatch's sub-gradient prior to its quantization. Akin to dithering and sigma-delta coding, this ensures that, at least in the limit, gradient values are exchanged at full precision; just distributed over time. This reduces bandwidth 32-fold.

In [17], this technique was applied to feed-forward DNNs, where it led to speed-ups of 7 when parallelizing over 20 GPUs (at a WER loss of about 0.1 percentage points). The dramatically higher ratio of computation to communication cost of windowed RNNs allows for a higher degree of parallelization. Table 1 shows that parallelization efficiency for 64 GPUs is still 55%, for a minibatch size of 8k, which the automatic minibatch-scaling process deems appropriate at about 6% into the training.

Most results in this paper used 16 GPUs, reducing the cost of processing 24h of data from about 18 hours to 90 to 100 minutes. This is the first time that 1-bit SGD was applied to RNNs and to sequence training. 1-bit SGD was a key enabler without which this paper would not have been feasible.

4. IMPROVING EFFICIENCY BY PREDICTING GROUPS OF FRAMES

Another key enabler for this technology is a small modification of the basic model, where we group multiple consecutive frames to share the same underlying RNN state. This makes decoding and sequence training more feasible.

4.1. Frame Grouping for Faster Decoding

The dramatically increased computation is not only an issue for training—it also hampers decoding. Consider that while GPUs can be assumed for training, they are typically not available in the data centers where speech recognition is deployed, due to cost and energy consumption/heat dissipation. The computational cost of our best model exceeds what can be computed in real-time on a single Intel multi-core server—a significant obstacle to real-world deployment.

We propose to augment the model such that *multiple consecutive frames* are predicted *jointly*, using the same underlying RNN state (and the same softmax parameters).

Consider a model that uses 20 context frames on either side of the predicted frame (denoted as 20+1+20). It requires to apply the model matrices 41 times for every frame.

If we, however, instead predict 8 consecutive frames from the 20 preceding and succeeding context frames (denoted as 20+8+20), we would reduce computation by $41/(48/8) = 6.8$ times. This brings computation cost back into a range of a single server (we can still process less concurrent streams per servers, but that is the price to pay for the improved accuracy).

Note that each of the 8 jointly predicted frames has a different left/right context length, ranging from (20+1+27) to (27+1+20). For example, the first of the 8 frames has a context of 27 frames to the right but only 20 to the left.

This structure has similarities with "context-sensitive chunk BPTT" in [19]. One difference is how we train it, by "jitter training" as explained next.

4.2. "Jitter Training"

While frame grouping accelerates decoding, it leads to less randomization of training samples, which we find to harm convergence. To address this, let's take the alternative viewpoint of individual frames in the training corpus. To them, the context "jitters," and the jitter *can be considered random*. This leads to a way to allow full sample randomization again in CE training *by simulating jitter*: (a) use

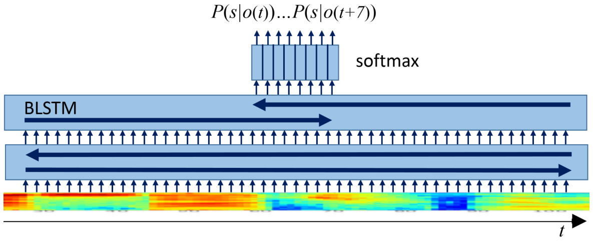

Fig. 1. *Illustration of reducing computation by frame grouping. 8 consecutive frames share the same underlying RNN state, where each frame sees a slightly differently shifted (jittering) context window. The window is advanced by 8 frames.*

an enlarged context window (27+8+27 in the above example) and (b) for every sample, simulate a pseudo-random jitter value (in our example in the range 0..7). To take advantage of GPU parallelism, this is implemented by applying the full enlarged context window but then interjecting code that zeroes out the respective hidden and error state values to simulate the jitter.

Figure 2 shows convergence when randomizing and jointly predicting groups of 8 consecutive samples vs. full sample randomization with jitter. The resulting word error rate after about 2300h of samples (jitter training has converged at this point) differs by over 1 percentage point.

4.3. Saving Memory to Enable Sequence Training

Another challenge with this model arises in sequence training: Memory consumption. Sequence training optimizes the model using a whole-sequence criterion that incorporates sequence constraints of the speech recognizer: the left-to-right HMM, dictionary, and language model [20, 3].

The problem is that as part of back-propagation of windowed RNNs, one needs memory to store activation and error vectors of *every frame* in *every sample* in the minibatch.

For example, in one of our best models, the activation state of one sample in one hidden layer consists of 512 LSTM cells × seven values per cell × 2 directions—for each of 41 frames in the window. Our implementaton then requires the same amount once again for the error state.

All in all, for a 6-hidden layer model, a typical utterance of 8 seconds requires 10.5 GB GPU RAM! Since there is also the output layer and the actual model, gradients, etc., even 8-second utterances are not feasible with the 12 GB of our modern NVidia K40m GPUs. Realistically, we'd have to discard utterances longer than 5 seconds.

Sharing RNN state across groups of frames reduces state memory accordingly. E.g. using groups of 8, we can now fit about 40 seconds in GPU RAM, which is enough for the large majority of utterances, and often allows to batch multiple utterances for efficiency.

5. EXPERIMENTAL RESULTS

5.1. Setup

We evaluate the windowed RNN on our research system for conversational speech transcription for Skype Translator. We chose this task because for speech-to-speech translation, where errors from recognition and translation compound, improving recognition accuracy is paramount even if that requires a significantly more expensive acoustic model.

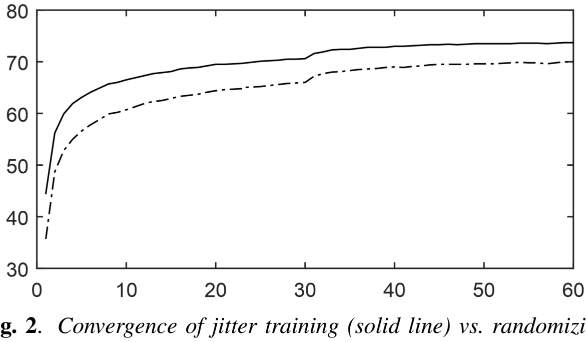

Fig. 2. *Convergence of jitter training (solid line) vs. randomizing consecutive groups of 8 frames. Shown is training-set frame accuracy over epochs of 24h.*

The training data consists of two publicly available Switchboard CTS corpora, 309 hours of SWBD1 and 1700h of Fisher; 1700h of subtitled in-house lectures recorded at Microsoft Research over many years; and 140h of TED lectures. Features are 40-channel wideband Mel-filterbank parameters using a 10-th root non-linearity instead of the usual log to gracefully handle zero filter outputs that occur for narrowband data. This does not degrade accuracy.

The baseline CD-DNN-HMM in this system is close to the one described in [2]. It has 7 hidden layers of dimension 2048 and an output dimension of 9000, a total of 46M model parameters. The 9000 senones were selected by a GMM system trained only on SWBD1/Fisher using narrow-band PLP features. The GMM alignments for the full 3850h corpus are also created with this narrowband SWBD1/Fisher model. All RNNs have a hidden dimension of 512 for each direction.

For testing, we use the publicly available Hub5'00 "sw" set (1831 utterances), the two parts of the RT03S corpus, as well as two in-house corpora. The first, denoted as "Tele-Conf.," consists of 1 hour of tele-conferences of 12 speakers. The utterances for each speaker were selected longest first until 5 minutes were reached, i.e. there is a bias for longer "content" utterances, as opposed to, e.g., back-channels. The second, denoted as "Speech Trans.," consists of domain data collected for developing Skype Translator, about 20k words.

Our compute servers are equipped with 8 NVidia Tesla K20Xm GPU cards each and connected through Infiniband.

The decoding language model is an interpolation of a trigram trained on the Fisher transcripts, and one trained on written background text. It has only about 8M trigrams, so we rescored the main results with larger LMs: a 5-gram trained on roughly 5 billion tokens from newswire, political proceedings, and web sources for the Tele-Conf and Speech-Trans corpora; and for the CTS sets an interpolated 4-gram obtained from three source-specific components, two trained with maximum-entropy estimation [21] on 4M words of Switchboard transcripts and 21M words of Fisher CTS transcripts, and one obtained from 191M words of conversational web data selected using frequent CTS n-grams, prepared by the University of Washington [22]. Recognition lattices were rescored with the combined LM, and new 1-best hypotheses were decoded, keeping the original LM weight at 13.25.

5.2. Comparing DNNs with windowed RNNs

Table 2 shows the WER improvement from using windowed RNN compared to our best DNN baseline system. Shown are recognition results for the 3850h training set on the 5 test sets, for both our best DNN and the "jitter-trained" 6 hidden-layer windowed bidirectional LSTM RNN (20 frames context on each side for a group of 8 consecutive frames).

After realignment and sequence training, a 28% relative WER reduction is achieved on the Hub5'00 test set. Gains are 16 and 17% on the two patts of RT03S, respectively, and 12 and 16% on our proprietary test sets. When training only using the CE criterion on GMM alignments, the gains are even larger, ranging from 18 to 33% (on Hub5'00).

For comparability, Table 3 shows the same experiments except the training set is limited to the public SWBD1/Fisher corpora. Comparing CE models, the gain on Hub5'00 is 27%; while on RT03S, we gain 20 and 17%. However, realignment and sequence training do not help here for the windowed RNN, reducing the gain on Hub5'00 to 16%. We believe we are seeing overfitting (while for the 3850h set, the more inhomogenous nature of the data has a regularizing effect).

Table 2. *Comparison between baseline DNNs and windowed RNNs using the 3850h training set, for five test sets. The numbers are word error rates (WER) in percent.*

configuration	Hub5'00	RT03S -fsh	RT03S -sw	Tele- Conf.	Speech Trans.
DNN	16.4	17.5	24.2	22.7	20.2
+ realign	15.3	16.3	22.7	21.9	19.9
+ sequence training	13.7	14.9	21.5	20.8	18.2
+ large LM	12.8	13.9	20.0	19.9	17.0
windowed jitter RNN	11.0	13.1	18.7	18.7	15.6
	(-33%)	(-25%)	(-23%)	(-18%)	(-23%)
+ realign	10.5	12.9	18.6	18.6	15.8
+ sequence training	9.9	12.5	17.8	18.1	15.1
+ large LM	9.2	11.5	16.7	17.5	14.2
	(-28%)	(-16%)	(-17%)	(-12%)	(-17%)

Table 4. *Impact of #layers and context window.*

windowing	Hub5'00	Tele-Conf.	Speech Trans.
DNN, 7 hidden layers			
10+1+10	16.2	22.8	20.4
15+1+15	16.4	22.7	20.2
20+1+20	16.5	23.2	20.5
windowed RNN, 4 hidden layers			
15+1+15	12.3	19.8	17.0
20+1+20	11.8	19.9	16.6
25+1+25	11.6	19.5	16.5
windowed RNN, 6 hidden layers			
15+1+15	11.7	19.6	16.5
20+1+20	11.1	19.5	16.0

5.3. Width and Depth

Table 4 shows the effect of the window width and number of hidden RNN layers. Unlike above, the RNN here does not use frame grouping or jitter training. While the DNN achieves the best results with 10 or 15 context frames on each side, the windowed RNN can take advantage of up to 25 frames. This is consistent with the hypothesis that the RNN is better able to normalize out time variability in the context.

5.4. Frame Grouping and "Jitter Training"

As discussed, frame grouping provides improved computational and memory efficiency, making decoding and sequence training feasible. Table 5 shows results for a context window of 20 on each side, while predicting a single frame (denoted as 20+1+20) vs. a group of 8 consecutive frames (20+8+20).

As expected from the convergence curve in Section 4, the reduced randomization due to grouping leads to notably worse accuracy, increasing WER by as much as 1.4 points for Hub5'00. The last row shows that by switching to "jitter training," the WER loss can be recovered. In fact, WER improves somewhat (up to 0.8 points), which we attribute to the increased context window. That increase is random in the range of 0 to 7 frames, but it seems the RNN is able to learn to still take advantage of the additional information when present.

5.5. 1-bit SGD Parallelization

Lastly, we have seen in Section 3 that 1-bit SGD parallelization scales well, better than with DNNs. Table 6 shows accuracies for different numbers of parallel nodes. The number of nodes affects the impact of quantization—less nodes lead to less averaging-out of quantization errors, while with more nodes, quantization is more prone to be dominated by outliers in the smaller sub-minibatches [17]. Indeed we find a sweet spot at 16 nodes, which is what we use throughout this paper, except for sequence training, where we used 8 since too small minibatches with utterance processing hurts load balancing. (The 8-GPU result in Table 6 did not finish in time, so we show a nearly converged result after 888 hours of data.)

In [17], it was shown that for DNNs, 1-bit SGD causes an accuracy loss of about 0.1 points when parallelizing over 10 nodes, compared to the unquantized, non-parallel baseline. Unfortunately, such a comparison is impractical for the windowed RNN due to training time—that number would take about 75 days to produce. We have to be content with the nevertheless large relative improvements over the DNN.

Lastly, in 1-bit SGD, the first epoch of 24h is typically not accelerated; we call this "warm start." However, with our windowed RNN, that first epoch already takes over half a day to compute, so we experimented with shorter warm-starts of 2.4 and 9.6 hours. However, the last two rows of table 6 show that even with 9.6 hours, we observe up to 0.3 points of WER loss, so all other experiments in this paper used the slow, full 24 hours warm start.

Table 3. *Comparison between baseline DNNs and windowed RNNs using the public SWBD1+Fisher training set.*

configuration	Hub5'00	RT03S-fsh	RT03S-sw
DNN	15.1	16.4	22.7
+ realign	13.9	15.2	21.6
+ sequence training	12.6	14.2	20.4
+ large LM	11.7	12.9	18.9
windowed jitter RNN	11.0	13.2	18.8
	(-27%)	(-20%)	(-17%)
+ realign	11.2	13.6	19.4
+ sequence training	10.6	13.2	18.9
+ large LM	9.9	12.3	17.8
	(-16%)	(-7%)	(-7%)

6. CONCLUSIONS

Like state-of-the-art feed-forward DNNs, windowed (truncated) RNNs operate on a finite sliding context window around the speech frame to predict, but their recurrent nature makes them resilient to timing variations and thus allows them to extract more class discriminant information from the window than a feedforward DNNs.

The windowed model allows online recognition for a bidirectional model, and to randomize training samples individually (as opposed to entire utterances), which leads to very fast and stable convergence.

We evaluated windowed bidirectional LSTM RNNs on conversational speech transcription. On the 2000h SWBD1/ Fisher corpus, the model reduced word errors on the Hub5'00 benchmark set by 16% relative over our best sequence-trained DNN. The model benefits overproportionally from more data: On an extended 3850h training set, the reduction on Hub5'00 is as large as 28%, reaching a WER of 9.2%.

The much increased computational and memory demands have been partially remedied by frame grouping, which makes decoding

Table 5. *Impact of frame grouping and jitter training.*

configuration	Hub5'00	Tele-Conf.	Speech Trans.
20+1+20	11.1	19.5	16.0
20+8+20	12.5	19.9	16.6
+ jitter training	11.0	18.7	15.6

Table 6. *WERs for different 1-bit SGD configurations.*

number of nodes (GPUs)	Hub5'00	Tele-Conf.	Speech Trans.
8 (after 888 hours)	11.5	19.3	16.3
16 (after 888 hours)	11.3	19.1	16.0
16	11.0	18.7	15.6
32	10.9	19.0	15.9
16, 2.4h warm start	11.3	19.5	16.1
16, 9.6h warm start	11.1	19.0	15.9

and sequence training feasible, while "jitter training" allows to still train these models with full frame randomization, averting an accuracy loss.

More sophisticated LSTM architectures like the mixture of softmaxes allow to extract even more discriminative information from spectral windows, and we believe that further gains are possible. Overall, we are closer than ever on our quest to reach human performance.

7. ACKNOWLEDGMENTS

The authors thank Donovan Smith and Jonathan Clark for rescoring with the Skype-Translator language model.

8. REFERENCES

[1] Dong Yu, Li Deng, and George E. Dahl, "Roles of pre-training and fine-tuning in context-dependent dbn-hmms for real-world speech recognition," in *NIPS 2010 workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

[2] Frank Seide, Gang Li, and Dong Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Interspeech 2011*.

[3] Hang Su, Gang Li, Dong Yu, and Frank Seide, "Error back propagation for sequence training of context-dependent deep networks for conversational speech transcription," in *ICASSP 2013*.

[4] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury, "Deep neural networks for acoustic modeling in speech recognition," *Signal Processing Magazine*, 2012.

[5] T. Robinson, M. Hochberg, and S. Renals, "The use of recurrent networks in continuous speech recognition," pp. 233–258. Kluwer Academic Publishers, 1996.

[6] Alex Graves, Abdel rahman Mohamed, and Geoffrey Hinton, "Speech recognition with deep recurrent neural networks," in *ICASSP 2013*.

[7] Hasim Sak, Andrew W. Senior, and Françoise Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *CoRR*, vol. abs/1402.1128, 2014.

[8] Mark Gales and Steve Young, "The application of hidden markov models in speech recognition," *Found. Trends Signal Process.*, vol. 1, no. 3, 2007.

[9] Herve A. Bourlard and Nelson Morgan, *Connectionist Speech Recognition: A Hybrid Approach*, Kluwer Academic Publishers, 1993.

[10] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," in *Advances in Neural Information Processing Systems 22*. 2009.

[11] Ossama Abdel-Hamid, Abdel-Rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, vol. 22, no. 10, pp. 1533–1545, Oct. 2014.

[12] Alex Graves, Navdeep Jaitly, and Abdel rahman Mohamed, "Hybrid speech recognition with deep bidirectional LSTM," in *ASRU 2013*.

[13] M. Schuster and K.K. Paliwal, "Bidirectional recurrent neural networks," *Trans. Sig. Proc.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[14] Li Deng, Ossama Abdel-Hamid, and Dong Yu, "A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, May 2013.

[15] George Saon, Hong-Kwang Jeff Kuo, Steven J. Rennie, and Michael Picheny, "The IBM 2015 english conversational telephone speech recognition system," *CoRR*, vol. abs/1505.05899, 2015.

[16] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[17] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs," in *INTERSPEECH 2014*.

[18] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu, "On parallelizability of stochastic gradient descent for speech DNNs," in *ICASSP 2014*.

[19] Kai Chen, Zhi-Jie Yan, and Qiang Huo, "Training deep bidirectional LSTM acoustic models for LVCSR by a context-sensitive-chunk BPTT approach," in *interspeech 2015*.

[20] Brian Kingsbury, "Lattice-based optimization of sequence classication criteria for neural-network acoustic modeling," in *icassp 2009*.

[21] Tanel Alumäe and Mikko Kurimo, "Efficient estimation of maximum entropy language models with N-gram features: An SRILM extension," in *interspeech 2012*.

[22] Ivan Bulyko, Mari Ostendorf, Manhung Siu, Tim Ng, Andreas Stolcke, and Özgür Çetin, "Web resources for language modeling in conversational speech recognition," *ACM Transactions on Speech and Language Processing*, vol. 5, no. 1, 2007.