LSTM-BASED ONE-PASS DECODER FOR LOW-LATENCY STREAMING

Javier Jorge, Adrià Giménez, Javier Iranzo-Sánchez, Joan Albert Silvestre-Cerdà, Jorge Civera, Albert Sanchis, Alfons Juan

Machine Learning and Language Processing (MLLP) research group
Valencian Research Institute for Artificial Intelligence, Universitat Politècnica de València (Spain)

ABSTRACT

Current state-of-the-art models based on Long-Short Term Memory (LSTM) networks have been extensively used in ASR to improve performance. However, using LSTMs under a streaming setup is not straightforward due to real-time constraints. In this paper we present a novel streaming decoder that includes a bidirectional LSTM acoustic model as well as an unidirectional LSTM language model to perform the decoding efficiently while keeping the performance comparable to that of an off-line setup. We perform a one-pass decoding using a sliding window scheme for a bidirectional LSTM acoustic model and an LSTM language model. This has been implemented and assessed under a pure streaming setup, and deployed into our production systems. We report WER and latency figures for the well-known LibriSpeech and TED-LIUM tasks, obtaining competitive WER results with low-latency responses.

Index Terms— automatic speech recognition, streaming, decoding, acoustic modeling, language modeling

1. INTRODUCTION

On-line or streaming automatic speech recognition (ASR) poses additional challenges to the off-line setup when state-of-the-art neural-based models are involved. The main practical challenge is that speech recognition must be performed under real-time constraints as the audio stream becomes available. This implies that the complete audio stream is not fully available when decoding is performed up to a certain point in time. This constraint, combined with the essential requirement of low latencies, is especially demanding when state-of-the-art, neural-based acoustic and language models are used.

On the one hand, acoustic modeling based on Bidirectional Long-Short Term Memory (BLSTM) networks has shown to be the current state-of-the-art in off-line ASR systems [1]. However, BLSTM network training requires to consider some future (right) context with respect to the current frame to compute its acoustic score. This means that the output of the ASR system must be delayed up to the point in which *enough* frames from the future have become available for decoding. In this sense, to facilitate streaming ASR, an in-depth study of using a sliding window that captures the left (past) and right (future) context with respect to the current frame was carried out in [2]. In brief, it was shown that BLSTM-based models outperform deep neural network (DNN) approaches in the streaming setup. This work was extended in [3] with a definition of a theoretical framework for the sliding window approach and a weighting scheme for overlapping frames.

On the other hand, language modeling based on LSTMs networks has become the dominant approach, providing the best results so far in many ASR tasks [4]. While these models are commonly used as part of the rescoring process using the n-best hypothesis or lattices [5, 6, 7], there is an increasing interest in providing an efficient way to integrate LSTM-based language models (LMs) into the decoding process, thus enabling their use in streaming ASR. Indeed, many approaches have been proposed in this regard, such as converting recurrent models into n-grams [8], using cache strategies [9], and reducing the computational complexity of the Softmax function [10]. It is worth noting, however, that these approaches were only focused on language modeling, that is, the decoding problem was not considered as a whole for (low-latency) streaming.

Recently, we proposed the use of LSTM-based LMs in the first pass of the decoding process, without any approximation or transformation in [11]. This is achieved by keeping the state of the LSTM LM network as part of the search structure, along with the Variance Regularization term [10] used to reduce the Softmax computational complexity. This allowed us to achieve a competitive performance while keeping the real time factor (RTF) below one. In this way, we paved the way to the use of LSTM LMs in a streaming setup, in which tight real-time constraints and low latency are critical. This is done in this work, that is, we extend the one-pass

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 761758 (X5gon);MCIU/AEI/FEDER,UE under the Multisub (RTI2018-094879-B-I00) research project and the Government of Spains FPU scholarship FPU18/04135.

decoding approach described in [11] by using LSTM LMs in a streaming setup. As in [3], acoustic modeling is based on BLSTMs, though in this work important requirements for streaming ASR are also considered, such as the normalization of the input features or the system architecture. More generally, this work can be seen as a thorough study of the effect that moving from an off-line to a streaming setup has on WER. It is shown that it highly depends on the feature normalization context and the availability of future acoustic context. In addition, given the tight real-time constraints of the streaming setup, detailed results on time latency are reported on reference tasks widely used in the literature such as LibriSpeech [12] and TED-LIUM [13].

2. STREAMING DECODER

2.1. One-pass decoder review

We followed the decoder structure proposed in [11]. The main aspects of this decoder will be commented briefly in this section. In this decoder, hypotheses are organized by their history, in a similar way as it is done in [14]. Therefore, the computation of the lookahead score should be carried out dynamically during decoding, involving several queries to the language model. To reduce the impact of that, in this approach it is proposed the use of the static lookahead tables, a structure that is precomputed in advance, providing the lookahead score efficiently during decoding. This structure is obtained from a pruned n-gram model, enabling the use of less memory during recognition.

On the other hand, when a word-end node is reached, we replaced the score that we got so far by the score from the LSTM LM. In order to compute this efficiently, we used the Variance Regularization term as a self-normalized function, avoiding the computation of the whole Softmax [10]. Along with this function, we followed a lazy strategy, when a language model node is created, to postpone the computation of the LM score as much as possible. A more detailed description of these steps can be found in [11].

We included two additional parameters in this one-pass decoder to control the WER/RTF trade-off: the Language Model Histogram Recombination (LMHR) and the Language Model Histogram Pruning (LMHP). The LMHR controls the size of the previous history considered to combine two hypotheses. The LMHP limits the number of new LM histories to be expanded, and thus the number of queries to the LSTM LM. As shown in [11], these parameters are really effective to reduce the RTF while keeping a good WER performance.

2.2. Streaming adaptation

Under streaming conditions, we cannot see the complete context for a given frame. Therefore, when using BLSTM networks for acoustic modeling, we have to assume a delay between the input and the output, that allows us to consider this temporal gap as a short-term context in order to obtain the acoustic score. In order to work with this acoustic lookahead, we have followed a similar strategy to [3]. It is based on the use of a sliding window over the sequence, where for each frame, we have a context of the $n_{\text{lookahead}}$ following seconds. These seconds, or equivalently, frames, were used in order to compute the forward and the backward steps over the sequence, obtaining a score for each frame within the window, using a BLSTM network as in [2]. We have done this frame by frame, meaning that there will be an overlap equal to the size of the window. Regarding the overlapping frames, we used a uniform weighted average of the acoustic scores to obtain the final score that will be provided to the decoder. In the extreme cases where the utterance is shorter than the window or the window goes beyond the sequence, we introduced zero padding up to the length of the window.

Regarding normalization of the features, we have included an initial delay that will be used to gather statistics to initialize the mean and the variance. We have a parameter, n_{norm}, that indicates the number of seconds that will be used to compute the statistics. Once these n_{norm} seconds have been accumulated, the normalization will be applied from the first frame to the last delayed frame, and then the recognition starts updating mean and variance frame by frame without including any additional delay. There is, indeed, an initial latency to obtain the first result from the decoder of n_{norm} seconds. However, in a real streaming setup, if this initial delay is small, it will not harm the global latency nor the performance of the system.

Our streaming setup follows a client-server architecture, where the server can process different requests at the same time. In order to manage that, in the server side, we have a pool of recognizers that will be paired with clients, serving requests in a separated way, while the recognizers share the same models, reducing the required CPU and GPU memory. This allowed us to accommodate different systems at the same time. This architecture is used in our TTP platform[1], currently available under registration, in English, Spanish and Catalan, and in the PoliSubs service[2] provided by the Universitat Politècnica de València.

3. EXPERIMENTS

3.1. Experimental setup

Table 1 provides some basic statistics of the corpora used to assess our approach: the LibriSpeech ASR corpus [12], and the third version of the TED-LIUM corpus [13]. Regarding the vocabulary, we have used the provided 200K words for LibriSpeech, and for TED-LIUM's we have selected 153K words. Regarding the partitions, we have used the **-other* for LibriSpeech and the **-legacy* ones for TED-LIUM.

[1] https://ttp.mllp.upv.es/

[2] https://apps.upv.es/ - https://polisubs.upv.es/

Table 1. Statistics of the corpora.

	LibriSpeech		TED-LIUM	
	Dur.(h)	Words	Dur.(h)	Words
Train	961	884M	452	258M
Dev	5.3	50K	1.59	17K
Test	5.1	52K	2.61	27K

Following the hybrid approach [15], we trained a context-dependent feed-forward DNN-HMM with three left-to-right states. We have obtained 8.3K and 10.8K tied states (senones) for LibriSpeech and TED-LIUM respectively after applying a phonetic decision tree [16], following a subphonetic modeling. To train our systems we have used the transLectures-UPV toolkit (TLK) [17].

We bootstrapped a bidirectional LSTM-HMM model using the previous DNN-HMM, as in [18]. For the BLSTM model, we used both TLK and TensorFlow [19], and an architecture of eight bidirectional hidden layers with 512 LSTM cells per layer and direction. Following [18], we performed chunking during training by considering a context to perform back propagation through time to a window size of 50 frames.

On the language modeling side, we used n-grams and LSTM LMs, combining them through linear interpolation. Apart from the 4-gram model provided for LibriSpeech, we trained a 4-gram Kneser-Ney smoothed LM for TED-LIUM, using the same data as in [13] and SRILM [20]. We obtained *OOV* ratios of 0.57% and 0.12% for LibriSpeech and TED-LIUM, respectively, on the dev sets. To compute the static lookahead tables, a pruned version of these n-gram models was computed.

The CUED-RNNLM toolkit [21] was used to train LSTM LMs. Noise Contrastive Estimation (NCE) criterion [22] was used to train faster, and the normalization constant learnt from training was used during recognition [23]. Based on the lowest perplexity models on dev, we selected the final models with 256-unit embedding layer and a hidden LSTM layer of 1024 units. Finally, the interpolation weights to combine the n-gram and the LSTM LM were $w_{ngram} = 0.15, w_{lstm} = 0.85$ for LibriSpeech and $w_{ngram} = 0.22, w_{lstm} = 0.78$ for TED-LIUM. Table 2 shows the perplexity on the dev partitions of these LM.

Table 2. Perplexity results on development partitions.

	LibriSpeech	TED-LIUM
4-gram	140.6	125.8
LSTM LM	86.2	88.0
LSTM LM + 4-gram int.	83.7	78.7

It is worth noting that we used the same models and pruning parameters for both the off-line and the streaming approach, in order to perform a fair comparison between the two setups. Following the experimental analysis in [11], we used the values of $LMHR = 10$ and $LMHP = 100$ for the one-pass decoder to provide best results in the development sets for both datasets. We carried out the computations related to both acoustic and language models in GPU, whereas the decoder was run in CPU, using a Intel Xeon(R) CPU E5-1620@3.50GHz with a GTX1080Ti with 12GB.

3.2. Assessment of normalization and lookahead contexts

This section is to assess the impact of the context for normalization (n_{norm}) and the impact of the lookahead context ($n_{\text{lookahead}}$) in terms of WER.

Table 3 shows the WER on the dev partition, for LibriSpeech and TED-LIUM, as a function of n_{norm} (in secs.). As discussed in Section 2.2, n_{norm} indicates the number of seconds that the system is allowed to compute statistics for feature normalization. For the results in this Table, we considered a value $n_{\text{lookahead}} = 0.5$ seconds, which is the value we used for chunking during training.

Table 3. Impact of normalization context on WER, on LibriSpeech and TED-LIUM.

n_{norm} (sec)	*LibriSpeech*	*TED-LIUM*
0	15.6	9.7
1	11.0	8.2
2	10.0	8.1
4	9.6	7.9
8	9.4	7.7
∞	9.4	7.6

From the results in Table 3, it is clear that normalization helps in improving performance. As expected, the best WER is achieved when the whole utterance is considered. Indeed, broadly speaking, with more information, the mean and variance are better estimated. However, it goes without saying an optimal value for n_{norm} should be in between extreme values. On our experience, an appropriate value for this parameter could be around 2 seconds. To us, this value is a sort of minimum for the system to collect meaningful statistics and respond after a reasonable waiting time.

Once this initial delay for the normalization is fixed to 2 seconds, we have performed an analysis of the impact on the WER of the parameter $n_{\text{lookahead}}$, which indicates the number of seconds of acoustic lookahead. Table 4 summarizes the results for these experiments, with the WER obtained for each corpus considering $\{0.125, 0.25, 0.5, 1, 2\}$ seconds of $n_{\text{lookahead}}$.

Results shown that the best WER is obtained with a 1 second and 2 seconds window length for LibriSpeech and TED-LIUM, respectively. While for TED-LIUM using the biggest considered size helped the performance, there is a degradation on WER for LibriSpeech, which could be due to the additional padding that we had to introduce for the small segments, very common in this dataset. In addition, results

Table 4. Impact of lookahead context on LibriSpeech and TED-LIUM on WER.

$n_{\text{lookahead}}$ (sec)	*LibriSpeech*	*TED-LIUM*
0.125	17.1	10.5
0.250	11.6	8.8
0.500	10.0	8.1
1.000	9.9	7.9
2.000	10.2	7.8

shown that there is an important gap in performance when using more than 0.250 seconds to compute the acoustic score, as a WER improvement of $\sim 14\%$ and $\sim 8\%$ could be achieved using 0.5 seconds to compute the sliding window, for LibriSpeech and TED-LIUM, respectively.

3.3. Latency assessment

Table 5 shows the average latency for LibriSpeech and TED-LIUM, as a function of $n_{\text{lookahead}}$ and n_{norm}. Here, latency refers to time elapsed between the end of the current hypothesis and the point in time at which it is actually delivered. So, for instance, if a hypothesis is delivered 2 seconds after its (acoustic) end, then the latency is 2 seconds. In contrast to Table 4, where WER is studied as a function of $n_{\text{lookahead}}$, here we focus on the average latency as a function of $n_{\text{lookahead}}$ and two values for n_{norm}, 2 and 0 seconds, with 0 seconds meaning that no time is devoted to gather statistics before recognition starts. While the setup with $n_{\text{norm}} = 2''$ was the one that we used to compute the WER in the previous Section, the results with $n_{\text{norm}} = 0''$ reflects the case in which sequences are long enough for normalization not to harm the global latency. Clearly, this gives an idea of how the decoder behaves in a real streaming setup. It is worth noting that these measurements were taken in the server side, not considering the network latency.

Table 5. Impact of lookahead context on LibriSpeech and TED-LIUM on the latency ($n_{\text{l}} = n_{\text{lookahead}}$, $n_{\text{n}} = n_{\text{norm}}$).

	LibriSpeech		*TED-LIUM*	
n_{l} (sec)	$n_{\text{n}} = 2''$	$n_{\text{n}} = 0''$	$n_{\text{n}} = 2''$	$n_{\text{n}} = 0''$
0.125	1.8 ± 0.5	0.6 ± 0.3	0.7 ± 0.5	0.3 ± 0.1
0.250	1.7 ± 0.5	0.6 ± 0.2	0.8 ± 0.4	0.5 ± 0.1
0.500	1.6 ± 0.5	0.8 ± 0.2	0.9 ± 0.3	0.8 ± 0.1
1.000	2.2 ± 0.5	1.4 ± 0.2	1.4 ± 0.2	1.3 ± 0.1
2.000	2.6 ± 0.6	2.9 ± 0.7	2.4 ± 0.3	2.3 ± 0.3

As can be seen in Table 5, the results for TED-LIUM are really good. In the TED-LIUM task, we get small latencies for both $n_{\text{norm}} = 0''$ and $n_{\text{norm}} = 2''$, up to $n_{\text{lookahead}} = 0.5$, from which we get reasonable WER figures. In the LibriSpeech task, however, a slight degradation of the average latency is observed, especially for $n_{\text{norm}} = 2''$. This is to the relative shorter in LibriSpeech ($\sim$6.5 seconds), as compared to TED-LIUM ($\sim$11.3 seconds), which results in the decoder not being able to fully recover from the initial delay. If we restrict ourselves to the realistic case of $n_{\text{norm}} = 0''$, then we can conclude that a good value for $n_{\text{lookahead}}$ is 0.5 seconds, which leads to a latency ~ 1 second and a competitive WER in LibriSpeech and TED-LIUM.

3.4. Off-line/Streaming comparison

With the values set for $n_{\text{norm}} = 2$ seconds and $n_{\text{lookahead}} = 0.5$ seconds, that involves a latency of $\sim$0.8 seconds, we have performed a final comparison between the off-line and the streaming systems, considering the *test* partitions. It is important to remark that, for the off-line setup, we have used the whole normalized sequence, while the streaming setup has just a limited context for both the acoustic features and normalization. Table 6 shows these results, reflecting how we can provide an streaming decoder with a similar performance in terms of WER, working with a low-latency setup.

Table 6. WER results on test sets for LibriSpeech and TED-LIUM.

	LibriSpeech	*TED-LIUM*
Off-line setup	10.2	8.2
Streaming setup	10.7	8.7

4. CONCLUSIONS AND FUTURE WORK

In this work we materialized the streaming decoder that was proposed in [11], using LSTM-based models for the acoustic and language modeling. We studied the impact of the normalization during the on-line decoding, as well as the impact of the acoustic future context in the WER and the latency. We obtained a system that provides a similar performance in terms of WER but working in a full streaming setup, with a latency of $\sim$1 second. We evaluated our approach in LibriSpeech and TED-LIUM, obtaining a competitive WER under a streaming regime.

As future work, we want to study the impact of using the same $n_{\text{lookahead}}$ window for training and decoding. Additionally, we want to consider alternative approaches for dealing with the limited future context, for example, considering all the past context or using faster models such as feed forward neural networks for the future context.

5. REFERENCES

[1] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[2] A. Mohamed, F. Seide, D. Yu, Jasha D., A. Stoicke, G. Zweig, and G. Penn, "Deep bi-directional recurrent networks over spectral windows," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 2015, pp. 78–83.

[3] A. Zeyer, R. Schlüter, and H. Ney, "Towards Online-Recognition with Deep Bidirectional LSTM Acoustic Models," in *Proc. Interspeech 2016*, 2016, pp. 3424–3428.

[4] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, "Exploring the limits of language modeling," *arXiv preprint arXiv:1602.02410*, 2016.

[5] X. Chen, X. Liu, A. Ragni, Y. Wang, and M. Gales, "Future word contexts in neural network language models," in *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2017, pp. 97–103.

[6] A. Ogawa, M. Delcroix, S. Karita, and T. Nakatani, "Rescoring n-best speech recognition list based on one-on-one hypothesis comparison using encoder-classifier model," in *Proc. ICASSP2018*. IEEE, 2018, pp. 6099–6103.

[7] S. Kombrink, T. Mikolov, M. Karafiát, and L. Burget, "Recurrent neural network based language modeling in meeting recognition," in *Twelfth annual conference of the international speech communication association*, 2011.

[8] E. Arısoy, S. Chen, B. Ramabhadran, and A. Sethy, "Converting neural network language models into back-off language models for efficient decoding in automatic speech recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 1, pp. 184–192, 2014.

[9] Z. Huang, G. Zweig, and B. Dumoulin, "Cache based recurrent neural network language model inference for first pass speech recognition," in *Proc. ICASSP2014*.

[10] Y. Shi, W. Zhang, M. Cai, and J. Liu, "Efficient one-pass decoding with nnlm for speech recognition," *IEEE Signal Processing Letters*, vol. 21, no. 4, pp. 377–381, 2014.

[11] J. Jorge, A. Giménez, J. Iranzo-Sánchez, J. Civera, A. Sanchis, and A. Juan, "Real-Time One-Pass Decoder for Speech Recognition Using LSTM Language Models," in *Proc. Interspeech 2019*, 2019, pp. 3820–3824.

[12] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *Proc. ICASSP2015*. IEEE, 2015, pp. 5206–5210.

[13] F. Hernandez, V. Nguyen, S. Ghannay, N. Tomashenko, and Y. Estève, "Ted-lium 3: twice as much data and corpus repartition for experiments on speaker adaptation," in *International Conference on Speech and Computer*. Springer, 2018, pp. 198–208.

[14] D. Nolden, *Progress in Decoding for Large Vocabulary Continuous Speech Recognition*, Ph.D. thesis, RWTH Aachen University, Germany, Apr. 2017.

[15] H. Bourlard and C. J. Wellekens, "Links between Markov models and multilayer perceptrons," in *Advances in Neural Information Processing Systems I*, D.S. Touretzky, Ed., pp. 502–510. Morgan Kaufmann, San Mateo, CA, USA, 1989.

[16] S. J. Young, J. J. Odell, and P. C. Woodland, "Tree-based state tying for high accuracy acoustic modelling," in *Proc. Workshop on Human Language Technology*, Plainsboro, NJ, USA, Mar. 1994, pp. 307–312.

[17] M.A. del Agua, A Giménez, N Serrano, Jesús Andrés-Ferrer, J Civera, A Sanchis, and A Juan, "The translectures-UPV toolkit," in *Advances in Speech and Language Technologies for Iberian Languages*, pp. 269–278. Nov. 2014.

[18] A. Zeyer, P. Doetsch, P. Voigtlaender, R. Schlüter, and H. Ney, "A comprehensive study of deep bidirectional LSTM RNNs for acoustic modeling in speech recognition," in *Proc. ICASSP2017*.

[19] M. Abadi, A. Agarwal, et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015, Software available from tensorflow.org.

[20] A. Stolcke, "SRILM – an extensible language modeling toolkit," Denver, CO, USA, Sept. 2002, pp. 901–904.

[21] X. Chen, X. Liu, Y. Qian, M. Gales, and P. Woodland, "CUED-RNNLM – An open-source toolkit for efficient training and evaluation of recurrent neural network language models," in *Proc. of ICASSP 2016*, Mar.

[22] A. Mnih and Y. Whye Teh, "A fast and simple algorithm for training neural probabilistic language models," *arXiv preprint arXiv:1206.6426*, 2012.

[23] X. Chen, X. Liu, M. Gales, and P. Woodland, "Improving the training and evaluation efficiency of recurrent neural network language models," in *Proc. of ICASSP 2016*, Brisbane, Australia, Apr. 2015, pp. 5401–5405.