A PRUNED RNNLM LATTICE-RESCORING ALGORITHM FOR AUTOMATIC SPEECH RECOGNITION

Hainan Xu[1,2]*, Tongfei Chen*[1]*, Dongji Gao*[1]*, Yiming Wang*[1]*, Ke Li*[1]*, Nagendra Goel*[3]*,*
Yishay Carmiel[2]*, Daniel Povey*[1]*, Sanjeev Khudanpur*[1]

[1]Center for Language and Speech Processing, Johns Hopkins University
[2]IntelligentWire, Seattle WA
[3]Go-Vivace Inc., USA
{hxu31,tongfei,dgao5,yiming.wang,kli26,dpovey1,khudanpur}@jhu.edu,
nagendra.goel@govivace.com, ycarmiel@intelligentwire.com

ABSTRACT

Lattice-rescoring is a common approach to take advantage of recurrent neural language models in ASR, where a word-lattice is generated from 1st-pass decoding and the lattice is then rescored with a neural model, and an n-gram approximation method is usually adopted to limit the search space. In this work, we describe a pruned lattice-rescoring algorithm for ASR, improving the n-gram approximation method. The pruned algorithm further limits the search space and uses heuristic search to pick better histories when expanding the lattice. Experiments show that the proposed algorithm achieves better ASR accuracies while running much faster than the standard algorithm. In particular, it brings a 4x speedup for lattice-rescoring with 4-gram approximation while giving better recognition accuracies than the standard algorithm.

Index Terms— automatic speech recognition, recurrent neural network language model, lattice-rescoring, heuristic search

1. INTRODUCTION

Language models are an essential component of *automatic speech recognition* (ASR) systems. In recent years, with the accessibility of greater computing power, *recurrent neural network language models* (RNNLM) [1] have become possible and have quickly surpassed back-off n-gram models [2] in various language-related tasks. However, because an RNNLM theoretically encodes infinite history lengths, it is virtually impossible to compile it to a static decoding graph; for this reason, RNNLMs are usually not directly used in decoding. The common method to take advantage of RNNLMs for ASR tasks is a 2-pass method: we decode on a pre-compiled decoding graph which is usually generated from a back-off n-gram language model as the first pass; instead of computing the 1-best hypothesis of the decoded results, we maintain a set of possible hypotheses and then in the second pass, use a more sophisticated neural-based model to rescore the hypotheses. N-best list rescoring and lattice-rescoring are among the most popular approaches.

In this work, we focus on lattice-rescoring, and propose a pruning-based lattice rescoring method, which utilizes a heuristic similar to that of A* search, greatly reducing the runtime of the algorithm. This heuristic search also helps picking better histories when expanding the lattices, resulting in better ASR performances as measured by *word-error-rate*s (WERs).

This project is developed as part of the open source speech recognition toolkit Kaldi [3]. During its development, we incorporated support for TensorFlow-based language models [4] into Kaldi, and we made the incorporation available to public. This will allow other Kaldi researchers to take advantage of TensorFlow developments, as well as TensorFlow researchers being able to evaluate their model in ASR tasks.

2. BACKGROUND

Many researchers have used or improved upon lattice-rescoring algorithms for ASR and related tasks. In [5], two methods are proposed to merge lattice-states in order to speedup lattice-rescoring and reduce sizes of the generated lattices; In [6], a refined pruning technique is used in order to reduce the search space when rescoring lattices with *long short-term memory* (LSTM) neural network language models. In [7], a novel neural network structure *succeeding word RNNLMs* (su-RNNLMs) is proposed and improvements in ASR tasks could be seen by lattice-rescoring. In [8], a pruned lattice-rescoring method is applied to neural machine translation.

THIS WORK WAS PARTIALLY SUPPORTED BY DARPA LORELEI AWARD NUMBER HR0011-15-2-0024, NSF GRANT NO CRI-1513128 AND IARPA MATERIAL AWARD NUMBER FA8650-17-C-9115 AND BY INTELLIGENTWIRE. THE AUTHORS WOULD ALSO LIKE TO THANK THE TENSORFLOW TEAM AT GOOGLE FOR THEIR HELP DURING THE PROJECT.

In [9], a lattice generation method based on [10] is used for on-the-fly lattice-rescoring to be used in real-time ASR applications.

Here we describe the baseline algorithm, i.e. the n-gram approximation lattice-rescoring method [5]. We describe the algorithm in the *weighted finite state transducer* (FST) framework commonly used in most of the major speech recognition systems [11].

2.1. Composition with LM-difference FST

Conceptually, RNNLM lattice-rescoring works by composing the lattice (call this A) with a finite state acceptor (call this B) that represents the *difference* between the n-gram FST that was already in the lattice, and the RNNLM, with costs suitably scaled if we are using an LM scale, so that by composing with B we instantaneously remove part of the existing LM cost and add in the new RNNLM cost. This FST B is generated on-demand by composing two deterministic finite state acceptors[1]: one for the backoff language model, and one for the RNNLM.

We note that in the baseline algorithm, the subtraction of the old LM weight and adding new LM weight could be done separately. However, performing them in a single rescoring pass keeps the costs in B close to zero, which makes it possible to adopt heuristics in the search which is necessary in the pruned algorithm described in Section 3.

2.2. Background: FST composition

We follow the basic FST composition algorithm to perform lattice-rescoring. The reader can refer to [12] for a more precise explanation, but the basic algorithm is as follows. Suppose we are composing $C = A \circ B$. Each state c in the output FST corresponds to a pair of states (a, b), and each arc in the output FST corresponds to a pair of arcs where the output label of the arc in A matches with the input label of the state in B (glossing over ϵ symbols and final-states). The algorithm can be implemented with a queue of state-pairs (a, b) for which arcs out of them need to be expanded. The queue discipline does not matter because we will eventually process all the state pairs that correspond to reachable states in the output.

2.3. An n-gram approximation method

An exact lattice-rescoring algorithm in practice is not feasible because the resulting lattice grows exponentially with respect to the depth of the original lattice. Usually an n-gram approximation algorithm is used to limit the size of B. The algorithm works by merging history states in B that are the same in the $(n-1)$ most recent words.

[1]Think of an acceptor as an FST where the input and output labels are the same, for current purposes

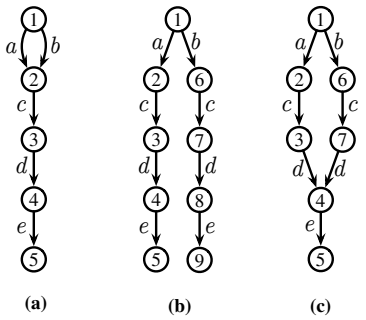

Fig. 1. Examples of lattices

For example, if the original lattice is the one shown in Figure 1.a, then the (exact) LM-difference FST B would have a topology shown in Figure 1.b, where each state corresponds to a unique history, and the weights associated with its arcs reflect the correct RNNLM weight computed from the correct histories. When using a 3-gram approximation, the LM-difference FST B' is shown in Figure 1.c, where state 4 and 8 in (b) are merged into a single state 4 (because they have the 2-word history `(c, d)`), and it encodes either history `a,c,d` or `b,c,d`. This choice is made arbitrarily during the composition algorithm depending on which sequence is first processed, and could potentially affect the computation of $p(e|c, d)$.

Using n-gram approximation allows much less computation and prevents the output lattice from exponentially exploding. However this speedup is achieved at a cost of accuracy – longer-than-n history words will not be exploited in RNNLM, and this results in worse performances.

3. PRUNED LATTICE-RESCORING ALGORITHM

Our proposed lattice-rescoring method is a kind of pruned FST composition with a priority queue based on a heuristic that allows us to expand the "probably best" paths first.

3.1. Pruned composition

What we are implementing here is partial, pruned composition, where not all possible arcs and states would be processed. We use a priority queue, based on a heuristic that is an approximation to the best cost we are likely to get from expanding a particular arc. The queue is, conceptually, a queue of arcs: that is, when we pop an element from the queue it corresponds to a state c (which can be mapped to a state-pair (a, b)), together with a particular arc leading from state a in the lattice that needs to be expanded.

In our application, A is an epsilon-free acyclic lattice and B is an epsilon-free, deterministic acceptor, which simplifies

certain things (it means, for instance, that there can be at most one arc leaving b that matches our arc leaving a).

3.2. Heuristic

In designing the heuristic, we assume that the cost contributions from B are small because B represents a difference of two language models. For the states in lattice A, we precompute "forward" and "backward" Viterbi costs for each state a. The forward cost $\alpha(a)$ is the cost of the best path from the start-state to a, and the backward cost $\beta(a)$ is the cost of the best path from a to a final-state (including the final cost).

We also define $\alpha(c)$ for a state c in the composed result C, as the cost of the best path from the start-state to c, and $\beta(c)$ as the cost of the best path from c to a final-state (including the final-cost), or $+\infty$ if there is no such path. For all these β quantities it is easy to get arc-level analogues of them (by limiting the paths to those leaving via a certain arc), but we will not develop specific notation for that. We will use the state-level notation for clarity, but the reader should understand that the queue is really on the individual-arc level.

Now, $\alpha(c) + \beta(c)$ will not be a very good heuristic at all because if we have not already expanded a state c, its $\beta(c)$ will always be $+\infty$. The same logic applies at the arc level. So if a state c corresponds to a pair (a, b), we use $H(c) = \alpha(c) + \beta(a)$ as the basic heuristic. The intuition here is that the extra costs from B are close to zero so $\beta(a)$ will be close to what $\beta(c)$ would be after expansion.

We next make a refinement. One concern is that for long utterances, if the RNNLM is systematically better than the baseline LM, the heuristic above would expand more states towards the end of the utterance than the beginning. So we make it

$$H(c) = \alpha(c) + \beta(a) + \delta(c) \quad (1)$$

where $\delta(c)$ is a correction factor that reflects how much we expect $\beta(c)$ to differ from $\beta(a)$. We compute $\delta(c)$ as follows: For any state c corresponding to a pair (a, b) where $\beta(c)$ is not infinity, we let $\delta(c) = \beta(c) - \beta(a)$. Otherwise we "borrow" the $\delta(c)$ from the previous state on the best path from the initial state to state c (we compute this in topological order, so this is always defined). As a special case, if the initial state has a $\beta(c)$ of infinity, we let its $\delta(c)$ value be zero.

3.3. Freshness

An exact implementation of the heuristic is not very efficient, because the computation $\alpha(c)$, $\beta(c)$ and $\delta(c)$ is quite costly. We use a work-around to solve this issue: most of the time we use stale values of these quantities that may be worse than (i.e. greater than) the exact value, and periodically we update all of them, using a schedule that doesn't affect the $O(n)$ runtime of the algorithm[2].

3.4. Termination

We terminate the algorithm based on a beam – Specifically, if the heuristic $H(c)$ is larger than the current best-path cost of the output lattice plus the beam, we neglect to process the state[3] c.

4. EVALUATION

We evaluate the algorithm for ASR tasks in 4 speech corpora, namely AMI-IHM, Switchboard (SWBD), Wall Street Journal (WSJ) and LibriSpeech (LIB). All the acoustic models are trained using the open source speech recognition toolkit Kaldi, with either TDNN [13] or lattice-free maximum mutual information (MMI) [14] models; Backstitch optimization method [15] is used during acoustic model training. The decoding is done on 3-gram language models, implemented with OpenFst [16], with explicit pronunciation and silence probability modeling as described in [17]. The RNNLMs are trained with TensorFlow [4], using the objective function described in [18] which prevents the need to compute a normalization term during test time scenarios. We compare the pruned lattice-rescoring algorithm with the n-gram approximation algorithm described in Section 2.3, denoted as "standard"; for the pruned lattice-rescoring experiments, we fix the beam size to be 6. For SWBD, speed perturbation [19] is used to further improve the acoustic model.

4.1. Rescoring Speed

Figure 2 compares the running speed for the 2 algorithms under different conditions. We report the average time it takes in rescoring all AMI lattices among 30 roughly equal sized jobs, and the numbers represent the average number of seconds to finish rescoring under different n-gram settings. We see that in all settings, the pruned algorithm runs faster than the standard algorithm. When comparing the same method in different n-gram settings, we notice that for the standard algorithm the runtime grows exponentially at a factor larger than 2 w.r.t. n, while for the pruned algorithm it grows much slower. In particular, the 4-gram pruned rescoring takes roughly the same time as 2-gram standard rescoring, and runs roughly 4 times faster than the 4-gram standard rescoring conterpart; for 5-gram rescoring, the pruned algorithm runs 10x faster.

4.2. Rescored Lattice Sizes

Figure 3 shows the average number of arcs per frame of the rescored AMI lattices under 2 different rescoring algorithms.

[2]Readers can find out more details of the algorithm at `src/lat/pruned-lattice-composition.{h,cc}` of Kaldi's repository

[3]We are speaking approximately here, because the algorithm is really at the arc level

Corpus	Test set	ARPA baseline	RNNLM rescoring with n-gram approximation					
			2-gram		3-gram		4-gram	
			standard	pruned	standard	pruned	standard	pruned
AMI-IHM (0.5)	dev	24.2	24.5	**24.0**	23.7	**23.4**	23.4	**23.3**
	eval	25.4	25.8	**25.0**	24.6	**24.4**	24.3	**24.2**
SWBD (0.8)	swbd	8.1	8.6	**8.2**	7.4	**7.2**	7.2	**7.1**
	eval2000	12.4	12.9	**12.3**	11.7	**11.5**	11.5	**11.3**
WSJ (0.8)	dev93	7.6	7.2	**6.9**	6.4	**6.2**	6.4	**6.2**
	eval92	5.1	4.6	**4.2**	4.1	**3.9**	3.9	**3.8**
LIB (0.5)	test-clean	6.0	5.5	**5.1**	4.9	**4.8**	4.8	**4.7**
	test-other	15.0	14.0	**13.2**	12.7	**12.4**	12.4	**12.3**
	dev-clean	5.7	5.0	**4.8**	4.4	**4.3**	**4.3**	**4.3**
	dev-other	14.5	13.7	**12.9**	12.3	**12.0**	11.9	**11.7**

Table 1. WER of Lattice-rescoring of Different RNNLMs

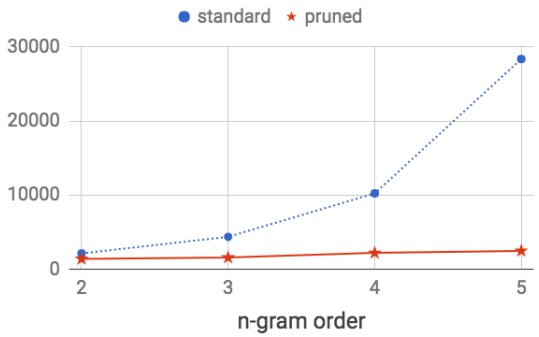

Fig. 2. Average run-time (in seconds) of lattice-rescoring

We can see that, using pruned lattice-rescoring algorithm allows for a smaller sized lattice to be generated, consistent with the results in Figure 2.

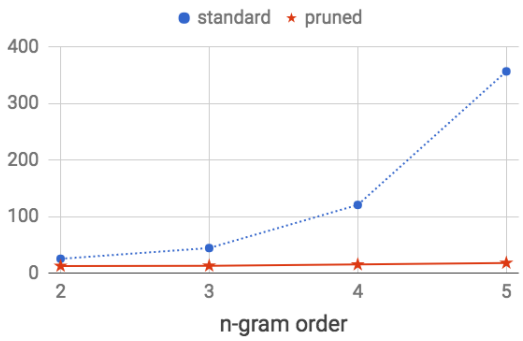

Fig. 3. Average number of arcs per frame of rescored lattices

4.3. WER performances

In Table 1 we compare the WERs of different rescoring results as well as the un-rescored baseline. The baseline is decoded from a 3-gram LM, during lattice-rescoring, the 3-gram model weight is interpolated with the computed RNNLM weight, and the interpolation weight is tuned per recipe (shown in the Corpus column in parenthesis). As we can see, in all recipes, our pruning method achieves better WERs than the standard algorithm for all n-gram orders. We notice for AMI, in the case of 2-gram, the standard rescoring does not bring any improvement, but the pruned algorithm brings a small gain over the un-rescored (3-gram) baseline. In particular, the gain is larger for smaller n-gram orders, where having longer correct histories brings more improvement to the system.

Combining the results in Table 1, Figures 2 and 3, we conclude that the proposed pruned lattice-rescoring algorithm runs much faster, greatly reduces the output lattice size, and improves ASR performances compared to the standard algorithm.

5. CONCLUSION

In this paper, we describe a pruned algorithm for performing lattice-rescoring with RNNLMs for speech recognition. Experiments show that the pruned algorithm runs up to 4 times faster than the standard algorithm in 4-gram approximation, and 10 times faster in 5-gram approximation. Because it follow a heuristic that first explores paths that are most likely to be the best path, the pruned algorithm allows an n-gram approximation algorithm to pick better histories in the lattice, and thus performs better than the unpruned algorithms.

6. REFERENCES

[1] Tomas Mikolov, Stefan Kombrink, Anoop Deoras, Lukar Burget, and Jan Cernocky, "Rnnlm-recurrent neural network language modeling toolkit," in *Proc. of the 2011 ASRU Workshop*, 2011, pp. 196–201.

[2] Joshua T Goodman, "A bit of progress in language modeling," *Computer Speech & Language*, vol. 15, no. 4, pp. 403–434, 2001.

[3] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al., "The kaldi speech recognition toolkit," in *IEEE 2011 workshop on automatic speech recognition and understanding*. IEEE Signal Processing Society, 2011, number EPFL-CONF-192584.

[4] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, Software available from tensorflow.org.

[5] Xunying Liu, Xie Chen, Yongqiang Wang, Mark JF Gales, and Philip C Woodland, "Two efficient lattice rescoring methods using recurrent neural network language models," *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 24, no. 8, pp. 1438–1449, 2016.

[6] Martin Sundermeyer, Zoltán Tüske, Ralf Schlüter, and Hermann Ney, "Lattice decoding and rescoring with long-span neural network language models," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[7] Xie Chen, Xunying Liu, Anton Ragni, Yu Wang, and Mark Gales, "Future word contexts in neural network language models," *arXiv preprint arXiv:1708.05592*, 2017.

[8] Huda Khayrallah, Gaurav Kumar, Kevin Duh, Matt Post, and Philipp Koehn, "Neural Lattice Search for Domain Adaptation in Machine Translation," in *Proceedings of the Eighth International Joint Conference on Natural Language Processing*, Taipei, Taiwan, 2017, Association for Computational Linguistics.

[9] Hasim Sak, Murat Saraclar, and Tunga Güngör, "On-the-fly lattice rescoring for real-time automatic speech recognition.," in *INTERSPEECH*, 2010, pp. 2450–2453.

[10] Andrej Ljolje, Fernando Pereira, and Michael Riley, "Efficient general lattice generation and rescoring," in *Sixth European Conference on Speech Communication and Technology*, 1999.

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

[12] Mehryar Mohri, Fernando Pereira, and Michael Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.

[13] Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur, "A time delay neural network architecture for efficient modeling of long temporal contexts," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[14] Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahremani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur, "Purely sequence-trained neural networks for asr based on lattice-free mmi.," in *INTERSPEECH*, 2016, pp. 2751–2755.

[15] Yiming Wang, Vijayaditya Peddinti, Hainan Xu, Xiaohui Zhang, Daniel Povey, and Sanjeev Khudanpur, "Backstitch: Counteracting finite-sample bias via negative steps," *Interspeech*, 2017.

[16] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri, "Openfst: A general and efficient weighted finite-state transducer library," *Implementation and Application of Automata*, pp. 11–23, 2007.

[17] Guoguo Chen, Hainan Xu, Minhua Wu, Daniel Povey, and Sanjeev Khudanpur, "Pronunciation and silence probability modeling for asr," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[18] Hainan Xu, Ke Li, Yiming Wang, Jian Wang, Shiyin Kang, Xie Chen, Daniel Povey, and Sanjeev Khudanpur, "Neural network language modeling with letter-based features and importance sampling," in *Acoustics, Speech and Signal Processing (ICASSP), 2018 IEEE International Conference on*. IEEE, 2018.

[19] Tom Ko, Vijayaditya Peddinti, Daniel Povey, Michael L Seltzer, and Sanjeev Khudanpur, "A study on data augmentation of reverberant speech for robust speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*. IEEE, 2017, pp. 5220–5224.