RNN-T lattice enhancement by grafting of pruned paths

Mirek Novak, Pavlos Papadopoulos

Amazon Alexa AI

{mieeknov, papavlos}@amazon.com

Abstract

Recurrent Neural Network Transducers (RNN-T) — a streaming variant of end-to-end models — became very popular in recent years. Since RNN-T networks condition the future output label sequence on all previous labels, the natural search space is represented by a tree. In contrast, hybrid systems employ limited-context language models, where the natural search space is a network, i.e. a lattice. While this lattice represents a rich set of hypotheses (n-best list), the search tree produced by RNN-T is more limited. In this work, we introduce a heuristic method which preserves some of the pruned RNN-T hypotheses by attaching them, or *grafting* them to the surviving hypotheses in the search tree. We achieved up to 21% oracle WERR without degrading the best-path WER. There is no impact on CPU cost, in fact we can speed up the decoder by using a smaller beam size, and still get improvements on oracle WER.

1. Introduction

In recent years, end-to-end (E2E) speech processing models have been the focus of research efforts and their success has led to the gradual replacement of traditional hybrid models [1]. A major advantage of E2E models over their hybrid counterparts is that they don't require explicit alignment of time frames and output labels, enabling them to produce output labels without consuming a time frame. This makes E2E models suitable for grapheme or word-pieces modeling, as well as deprecating the need of an explicit pronunciation lexicon.

Of particular interest are Recurrent Neural Network Transducers (RNN-T) [2], E2E models that are suitable for real-time streaming applications. As described in [3], assume an input of feature vectors (e.g. MFCCs, filterbanks, embeddings, etc) $\mathbf{x} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$ and a corresponding label sequence $\mathbf{y} = [y_1, \ldots, y_U]$, with $y_u \in \mathcal{Y}$ where $\mathcal{Y}$ is the set of output labels, e.g. word pieces [4]. Following, we can define a set of alignments $\mathcal{B}(\mathbf{x}, \mathbf{y})$ between the frame-level feature vectors and all the possible label sequences $\hat{\mathbf{y}}$ such that $\hat{\mathbf{y}} = [\hat{y}_1, \ldots, \hat{y}_{T+U}]$ and $\hat{y}_i \in \mathcal{Y} \cap \{\langle \mathrm{b} \rangle\}$ where $\langle \mathrm{b} \rangle$ is a blank symbol, and $\hat{\mathbf{y}} = \mathbf{y}$ after the blank symbols are removed from $\hat{\mathbf{y}}$.

Then the distribution of output symbols sequence is:

$$\begin{aligned} P(\mathbf{y}|\mathbf{x}) &= \sum_{\hat{\mathbf{y}} \in \mathcal{B}(\mathbf{x},\mathbf{y})} P(\hat{\mathbf{y}}|\mathbf{x}) \\ &= \sum_{\hat{\mathbf{y}} \in \mathcal{B}(\mathbf{x},\mathbf{y})} \prod_{i=1}^{T+U} P(\hat{y}_i|\mathbf{x}_1^{t_i}, y_0^{u_i}) \end{aligned} \tag{1}$$

where $\mathbf{x}_1^{t_i}$, $y_0^{u_i}$ are the sequences of input feature vectors and output labels respectively, i.e. $\mathbf{x}_1^{t_i} = \mathbf{x}_1, \ldots, \mathbf{x}_{t_i}$ and $y_0^{u_i} = y_0, \ldots, y_{u_i}$ with y_0 being a special "beginning of sentence" symbol. Notice that u_i and t_i denote the number of non-blank and blank symbols respectively in the partial alignment sequence $\hat{y}_1, \ldots, \hat{y}_i$.

Since the predicted label $\hat{y}_i$ depends on the complete history $[y_1, y_2, \ldots, y_{u_i}]$, a tree search with exponential complexity is needed to guarantee finding the best path. Such a search is not tractable for longer utterances, since it would need to keep history for all output labels in each time step. Instead, a beam search approach [5] is typically employed. This beam search method trades memory for accuracy and uses two parameters, i.e. *beam size*, that limits the number of hypotheses extended in any given time frame and *beam delta cost*, which limits the cost (negative log probability) of the expanded hypotheses to fall within a certain threshold Δ of the best scoring path. Thus, we prune some paths during inference, and the output tree of RNN-T beam search contains at most as many hypotheses as the *beam size* allows. In addition to that, RNN-T networks are usually decoded with a small number of *beam size* for efficiency [1]. Due to the combination of using a small *beam size* and pruning paths in the search tree, RNN-T lattices don't contain as much information as the lattices of hybrid systems, which have a network structure.

However, there are many downstream technologies that could benefit from lattices that contain meaningful information. For example, the authors in [6] encode lattices to train an attention encoder-decoder model, while in [7] the authors detect keywords in audio streams through the lattices which is especially useful when dealing with low quality recognition outputs, e.g. due to noise conditions. Furthermore, RNN-T suffer from a decrease in performance when encountering out of domain or tail cases (e.g. contact names) [8] . Rescoring mechanisms are usually employed to alleviate this issue. These rescoring methods fall into two major categories, first-pass methods such as shallow fusion [9] which influence the output label generation, and second-pass methods that operate on the generated hypotheses (e.g. the n-best or the lattice). Usually, second-pass rescoring utilizes external language sources (text data that were not in the utterances we trained the RNN-T) to achieve better results. Of course, for the second-pass rescoring methods to have any effect on WER, the generated hypotheses need to include a hypothesis close to the correct utterance, or in other words exist as path in the lattice, which is something we can measure by utilizing the oracle WER metric, i.e. the hypothesis in the lattice that gives the lowest WER [10].

The focus of this work is producing lattices from an RNN-T model which contain rich information without exponentially increasing the search cost. To that end we propose a novel heuristic method, by merging — or "grafting"— pruned branches to better scoring branches, ensuring that the newly generated paths will have higher cost than the best path, i.e. produce an enhanced lattice without altering the best path decision.

The rest of the paper is organized as follows. In Section 2 we introduce the main idea behind grafting. In Section 3 we describe the grafting algorithm, and in Section 4 we will present and discuss our findings Finally, in section 5 we will draw our conclusions.

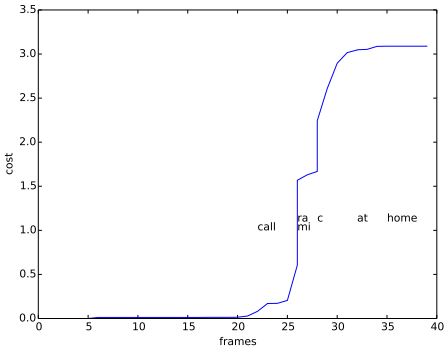


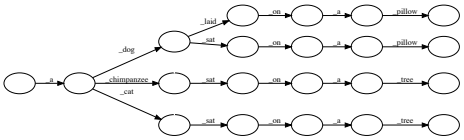
Figure 1: *Cumulative cost of the best path for "call Mirek at home" utterance.*

2. Motivation for Grafting

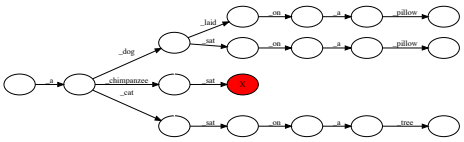
The posterior distributions produced by the joint network of the RNN-T decoder tend to be sharp, in fact a greedy search often provides the best scoring path. However, this is true only when there is good match between training and testing data, i.e. testing data do not contain instances not seen in training. In those cases, the cumulative cost of the best path is a monotonically increasing function with low first derivative. On the other hand, when there is a mismatch (e.g. inferring a word which rarely appears in the training data), a large cost can be accumulated within a few frames. An example is shown in Figure 1 for an utterance "*call Mirek at home*" containing a rare proper noun. The Figure shows a large increase of the cost function in the region of the proper noun (frames 25 to 31), while it is mostly flat otherwise. Obviously, the search fails to find the correct answer. We could increase the beam size when encountering uncertain regions to explore more paths, however those are likely to be pruned off in the future due to their inherent high cost.

In [3] this problem is addressed to some extent by merging contexts of certain length. This is similar to limiting the language model history (n-grams) in hybrid systems. The authors have tested their method on many datasets, and report that there is no impact on WER when the model was trained with the limited context (n-gram), but there is a small degradation when the context limitation is applied only during inference. Merging the paths which are likely to have the same future effectively makes room for other paths to survive until the completion of the search.

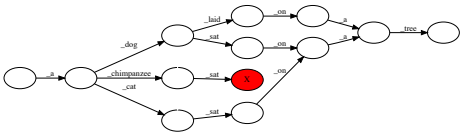
We propose an alternative method for improving the oracle WER. The difference between n-gram history based merging and grafting is shown in Figure 2. Figure 2a shows the tree search space of an utterance with *beam size* of 4. Assume that one of the paths is pruned off because a tighter beam size is used as depicted in Figure 2b, and that the resulting paths are ordered by cost, i.e. the top branch of the tree has higher cost (lower probability) and the bottom branch has lower cost (higher probability). The application of bigram merging [3] results in the tree of Figure 2c. We merge the "sat on" parts of the "a dog sat on a pillow" path with the "a cat sat on a tree" path, and then we merge the "on a" part of the "a dog laid on a pillow" path with the result of the previous merging operation. Notice that this procedure creates a generalization that produces an invalid hypothesis "dog sat on a tree". This is because the bigram history is too short to capture the longer dependency between individual animals and their seating preferences. Furthermore, the previously pruned path still remains pruned, since the search



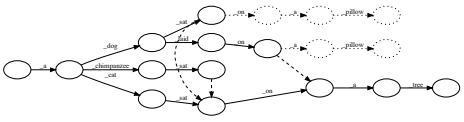
(a) *Full search tree.*



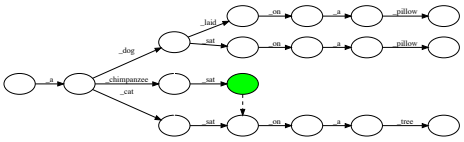
(b) *Pruned search tree.*



(c) *2-gram merging.*



(d) *Tentative 1-gram merging.*



(e) *Grafting.*

Figure 2: *Example of a search tree (a) and its pruned version (b). Examples of bi-gram and uni-gram merging are shown in (c) and (d) respectively. In (e) we show how the tree looks after grafting*

space reducing recombination happens later in the time.

A unigram merging approach, as shown in Figure 2d could alleviate some of the problems the tighter beam and the bigram merging introduced. For example we could preserve the pruned path of the *chimpanzee* sitting preference. However, this would still create invalid hypotheses regarding the sitting and laying preference of the *dog*. To overcome this issue we could modify unigram merging to include **only** the paths that are pruned, as shown in 2e. This way the surviving paths could be extended with full history and at the same time provide a mechanism that preserves the information that would be lost due to pruning. Indeed, this is main idea behind grafting – a pruned path is *grafted* to a better scoring path. Since the grafted path has always higher cost and the main path is still extended with full history, the cost of the best path remains the same.

To summarize: in the tree search, the pruned paths are lost unless we relax the search beam to preserve them; in the n-gram merging the paths are always recombined and the model needs to be re-trained get the full benefit; in grafting we perform a selective recombination which does not impact the search.

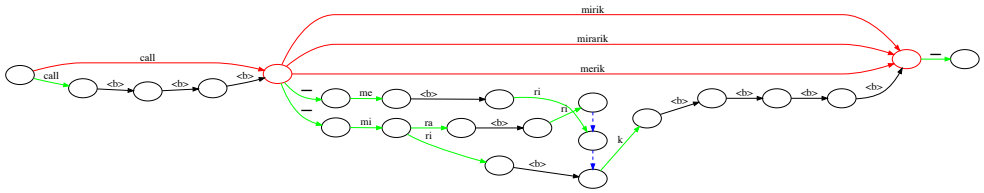


Figure 3: *Examples of lattice link creation*

3. Algorithm Description

A typical inference algorithm uses token search, where the token is a tuple *(cost,predState,label,parent)*. The *cost* stores the accumulative cost of the hypothesis, *predState* encodes the state of the predictor network of the RNN-T represented by unique integer id, *label* is the current $\hat{y}_i$ and *parent* is a pointer to the previous token, used to back-traverse the hypothesis. Implementation of grafting for such algorithms requires relatively small changes. We extend the token structure to *(cost,predState,label,parent,sibling)*, where *sibling* is used to attach the grafted path. When multiple paths are being grafted to a particular token, the *sibling* of that token points to a linked list of those paths.

Compared to the original Graves's algorithm [2], we change the way the tokens are discarded when they fall outside of active search beams. We first check if the token can be grafted and discard it only when the grafting fails (i.e. there is no alive hypothesis satisfying the unigram history).

The original task of trellis-tree conversion to lattice or n-best is trivial. The resulting n-best list will contain as many entries as there are final tokens. Tracing back the tokens, the costs and labels are combined into word lattice link cost and word string (new word spellings are dynamically created from the token labels representing word pieces).

However, when decoding with grafting, the situation becomes more complex. There can be many more n-best paths than the number of final states and exhaustive enumeration would be expensive and often unnecessary. To address this issue we first convert the trellis into a word lattice (which will be a network rather than a tree) and than use forward-backward algorithm to produce n-best of the desired size. Finally, we apply length score normalization.

The lattice creation algorithm for grafting processes a set Q of tuples $(token, linkCost, string, linkEndState)$ representing partially created lattice links, where *linkCost* is the partially accumulated cost of the next lattice link, *string* is a partially concatenated spelling of the words associated with the link (e.g. a sequence of word pieces), and *linkEndState* is the lattice state in which this link ends.

The algorithm starts by inserting one tuple for each final state in Q, then we remove one tuple at a time from the set, and it gets updated it by traversing the parent of the token. If the linked list attached by the sibling pointer is non-empty, a new tuple is created for each entry in the list. Once a word delimiter is encountered, a new lattice link together with a new state (if needed) is created. Pseudocode of the decoding procedure with grafting is given in Algorithm 1. Figure 3 shows some examples of new word lattice links (in red) created using Algorithm 1 in the top half, as well as the corresponding trellis in the bottom half. The sibling connections are in blue dashed lines, while the

Algorithm 1 Grafting Algorithm

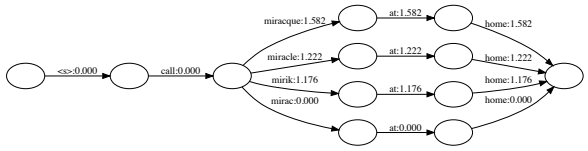
$F = \{(finalToken, 0.0)\}; Q = \{\}$
$numStates = 0; Lattice = \{\}; LatStateMap = []$
for $(tok, linkCost) \in F$ **do**
  $(tok, linkCost, < empty >, numStates) \rightarrow Q$
end for
while $Q \neq \{\}$ **do**
  $Q \rightarrow (tok, linkCost, string, linkEndState)$
  $linkCost+ = tok.cost - tok.parent.cost$
  $string = tok.label + string$
  if $tok.sibling \neq NULL$ **then**
    $(tok.sibling, linkCost, string, linkEndState) \rightarrow Q$
  end if
  if $tok.label == wordDelimiter$ **then**
    if $tok.parent \in LatticeStateMap$ **then**
      $linkStartState = LatStateMap[tok.parent]$
      $(linkStartState, linkEndState, string, linkCost) \rightarrow Lattice$
    else
      $start\_state = NumStates$
      $LatStateMap[tok.parent] = NumStates$
      $incr(NumStates)$
      $(linkStartState, linkEndState, string, linkCost) \rightarrow Lattice$
      $linkCost = 0.0$
      $string =< empty >$
      $tok = tok.parent$
    end if
  else
    $tok = tok.parent$
  end if
end while

label emitting and blank arcs are green and black lines respectively. Moreover, Figure 4 shows the complete word lattice for an utterance with and without grafting. In both cases the arc scores represent the distance of each link from the best path.

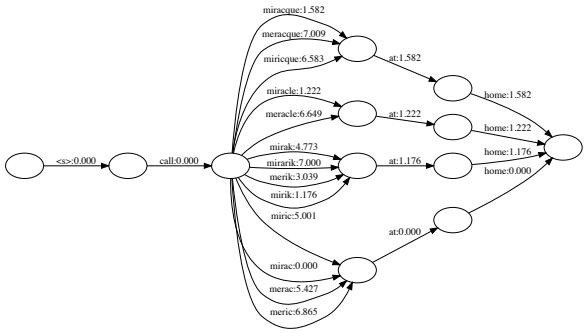
The CPU cost of the algorithm is negligible for the search itself (creation of the link lists). The cost of the lattice generation increases linearly with the number of links and thus is higher for grafting. Nevertheless, it is less than 0.5% of the total decoding time.

4. Results and Discussion

In order to measure the efficacy of our method we need to determine if the lattice contains meaningful information. To that end, we compare how the oracle WER behaves with regard to the size of the n-best list [10], when grafting is enabled versus the baseline case where it is not. Although we have tried our method in various datasets and observe consistent improvements, we focus our attention on a dataset of "tail" utterances dominated with proper nouns, specifically by first names (e.g. Mirek, Pavlos, etc) due to size limitations. This dataset contains de-identified user data from a commercial voice service and consists of approximately 200 hours from 21000 different speakers. Moreover, our decoding model is an RNN-T network of about 55 million parameters, and none of the utterances or the speakers of our testset were used in its training. First pass re-



(a) *Without grafting,*



(b) *with grafting.*

Figure 4: *Word latttice example for utterance* "call mirek at home". Best path is "call mirac at home".

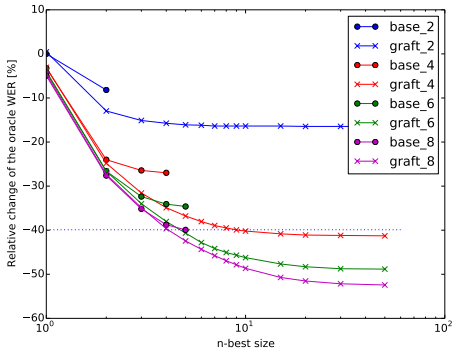


Figure 5: *Oracle WER change relative to the best path WER at beam size 2 for different sizes of n-best list. These results were produced from first pass only, no second pass rescoring was applied. Base and graft stand for baseline and grafting respectively.*

sults are the output of the network without any rescoring, while in the second pass the output hypotheses are rescored as described in [8]. Finally, we limit the generated paths when grafting is enabled to 50 for all the experiments. This is different than increasing the size of the beam search to 50 since it would exponentially increase the computational cost.

In Figure 5 we present the oracle WER (wer_O) change relative to the WER of best path when using a beam size of 2 (wer_{B_2}), calculated as $\frac{wer_O - wer_{B_2}}{wer_{B_2}} \times 100$. These results were produced by using only first pass, and no rescoring was applied. As expected, the depth of the n-best list is limited by the beam size for each of the baseline cases, while in their grafting counterparts it is deeper. Furthermore, we notice that for each beam size the oracle WER improves whenever we apply grafting.

In Table 1 we present the relative change of oracle WER when grafting is applied ($wer^{(m)}_{O_b}$) with respect to the baseline oracle WER ($wer^{(m)}_{O_b}$), calculated as $\frac{wer^{(m)}_{O_g} - wer^{(m)}_{O_b}}{wer^{(m)}_{O_b}} \times 100$ **for each beam size** m. These results show that the lattices generated using grafting contain valuable information and are able to capture the contents of an utterance more accurately than their baseline counterparts, something that holds true even when we use a small value for beam search, i.e. the lattice of the baseline case would contain at most 2 hypotheses. This observation could be essential in guiding the design of rescoring methods.

Table 1: *Grafting oracle WER change relative to the baseline oracle WER for different beam sizes*

beam size	2	4	6	8
relative WER change	-9.01%	-19.58%	-21.73%	-20.81%

Next, we study the effect grafting has on WER when using second pass rescoring and present these results in Table 2. We measure the change of best path WER with grafting relative to the best path WER without grafting for different beam sizes. We have not tuned any of the rescoring components with grafting lattices. For each beam size we observe improvements in WER when grafting is applied, this happens because the rescoring mechanisms have more hypotheses to work with, and grafting enhances the lattice by retaining meaningful information. These results indicate that grafting can be a method to boost the performance of ASR systems without involving expensive operations (e.g. fine tuning a model).

Table 2: *Grafting vs baseline relative change of best path WER for different beam sizes after second pass rescoring*

beam size	2	4	6	8
relative WER change	-5.52%	-2.81%	-2.92%	-2.56%

Finally, we measure the change of best path WER with grafting relative to the best path WER of beam size 8 without grafting (our best performing baseline) after second pass rescoring, and present the resuls in Table 3. We notice that with grafting we can reduce the beam size up to a point and still get an improvement – when using grafting with beam size 6 there is a small improvement over the baseline case of beam size 8. This means that grafting compensates for the lost paths in the baseline system due to reduced beam size. We could reduce the beam size (and improve the speed) with no cost on WER.

Table 3: *Change of best path WER with grafting relative to the baseline case of beam size 8 after second pass rescoring*

beam size	2	4	6	8
relative WER change	5.76%	2.77%	-0.6%	-2.56%

5. Conclusions

We presented a novel method, that enhances RNN-T lattices by maintaining useful information, and demonstrated its validity by showcasing improvements in the oracle WER of first-pass decodings as well as how it can improve WER when we use second-pass rescoring. These improvements do not require to increase the beam search size, nor do they depend on model retraining. Grafting has no impact on the computational cost, except for a negligible increase in the lattice generation. In fact, the search beam can be tightened (reducing the computational cost), and still get improvements on oracle WER and second-pass WER.

Acknowledgment: We thank Jasha Droppo for his advice and guidance in the execution and publication of this work.

6. References

[1] T. N. Sainath, Y. He *et al.*, "A Streaming On-Device End-To-End Model Surpassing Server-Side Conventional Model Quality and Latency," in *Proc. of ICASSP*, 2020.

[2] A. Graves, "Sequence transduction with recurrent neural networks," in *International Conference on Machine Learning: Representation Learning Workshop*, 2012.

[3] R. Prabhavalkar, Y. He, D. Rybach, S. Campbell, A. Narayanan, T. Strohman, and T. N. Sainath, "Less is more: Improved RNN-T decoding using limited label context and path merging," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 5659–5663.

[4] M. Schuster and K. Nakajima, "Japanese and Korean voice search," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 5149–5152.

[5] M. Jain, K. Schubert, J. Mahadeokar, C. Yeh, K. Kalgaonkar, A. Sriram, C. Fuegen, and M. L. Seltzer, "RNN-T for latency controlled ASR with improved beam search," *CoRR*, vol. abs/1911.01629, 2019. [Online]. Available: http://arxiv.org/abs/1911.01629

[6] P. Pandey, S. D. Torres, A. O. Bayer, A. Gandhe, and V. Leutnant, "Lattention: Lattice-attention in asr rescoring," To appear in ICASSP 2022.

[7] D. Can and M. Saraclar, "Lattice indexing for spoken term detection," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 8, pp. 2338–2347, 2011.

[8] A. Gourav, L. Liu, A. Gandhe, Y. Gu, G. Lan, X. Huang, S. Kalmane, G. Tiwari, D. Filimonov, A. Rastrow, A. Stolcke, and I. Bulyko, "Personalization strategies for end-to-end speech recognition systems," in *IEEE International Conference on Acoustics, Speech and Signal Processing, (ICASSP)*, 2021, pp. 7348–7352.

[9] D. Le, G. Keren, J. Chan, J. Mahadeokar, C. Fuegen, and M. L. Seltzer, "Deep shallow fusion for rnn-t personalization," *2021 IEEE Spoken Language Technology Workshop (SLT)*, pp. 251–257, 2021.

[10] L. Rodriguez, A. Reddy, and R. Rose, "Efficient integration of translation and speech models in dictation based machine aided human translation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 4949–4952.