INTERSPEECH 2023
20-24 August 2023, Dublin, Ireland

Prefix Search Decoding for RNN Transducers

Kiran Praveen, Advait Vinay Dhopeshwarkar, Abhishek Pandey, Balaji Radhakrishnan

Samsung R&D Institute Bangalore, India

{k.praveen.t, a.dhopeshwar, abhi3.pandey, balaji.r}@samsung.com

Abstract

Automatic Speech Recognition (ASR) has seen a surge in popularity for Recurrent Neural Network Transducers (RNN-T) in recent years and shows much promise. RNN-Ts were introduced as an extension of Connectionist Temporal Classification (CTC) models. While CTC models have prefix search as the widely used decoding strategy, it appears to have been overlooked in favour of other decoding strategies such as time-synchronous decoding (TSD) and alignment-synchronous decoding. In this work, we introduce prefix search decoding, looking at all prefixes in the decode lattice to score a candidate. We show that our technique aligns more closely to the training objective compared to the existing strategies. We compare our technique with the originally proposed TSD, using Librispeech and AMI-IHM datasets. We find that while prefix search is closer to the training objective, with larger datasets the performance improves significantly, while with lower size datasets the performance degrades.

Index Terms: speech recognition, rnn-transducers, prefix search decoding

1. Introduction

Automatic Speech Recognition (ASR) has progressed from having Gaussian Mixture Model (GMM) - Hidden Markov Model (HMM) based systems, to DNN-HMM [1] based systems, then to fully end-to-end based systems like Connectionist Temporal Classification (CTC) [2] loss based networks and recently to sequence-to-sequence networks which make use of an encoder-decoder based auto-regressive inference approach. RNN-T [3] is one such end-to-end based system which was built on top of the ground-works of CTC, addressing a few short-comings of it like conditional independence assumption and target sequences needing to be shorter than the input sequence. RNN-Ts have been gaining traction in the past few years owing to it's competitive performance when compared to other models.

The idea of the RNN-T training is to maximize the probability of all possible alignments of a target sequence with that of an input sequence. The idea of creating the lattice for the alignments and finding all possible alignments is well formulated and explained in the original work. When it comes to inferring the output sequence for a new utterance after training, there are multiple approaches, the main approach being time-synchronous decoding. However, when compared to CTC, which RNN-Ts were expanded from, the latter seems to be missing the main decoding scheme used for decoding an output from a CTC lattice, which is prefix-search decoding. There have been other decoding strategies introduced like alignment-length synchronous decoding (ALSD) [4], but we haven't come across any work which tries to find the most probable output sequence by considering all possible prefixes of the sequence, which is the very objective that the loss minimization tries to achieve. This work aims to expand and modify prefix-search decoding to work with RNN-T networks, as it seems to have been overlooked as a decoding strategy. In this work, we introduce and formulate this decoding strategy and compare it with the existing TSD strategy. We investigate the advantages and shortcomings of the proposed decoding strategy and present our observations.

The paper is structured as follows, Section 2 discusses the relevant literature related to this work. In Section 3 we explain how RNN-Ts work and explain the existing TSD algorithm. In section 4 the proposed prefix-search decoding strategy for RNN-Ts are explained and the algorithm is presented. Section 6 explains the details on how we did all our experiments, and in Section 7 we present our results. Finally, discussion and summary presented in Sections 8 and 9.

2. Related work

With the introduction of RNN-T in [3], the author proposes an algorithm to get the best output sequence based on time-synchronous beam search. This method scales to long sequences and targets the low computational cost by trading off against the search accuracy. Later, people have worked on different strategies with the major focus being either optimization of existing used algorithms or suggested new search algorithms.

Jain et al. [5] improved TSD by pruning out unlikely paths during decode by comparing the scores of a set of hypotheses present at time t and $t + 1$ and improved decoding latency. Tripathi et al. [6] introduces a breadth-first search algorithm in which during the search they also try to approximate the summation for all possible alignment paths in training.

Alignment-length synchronous decoding (ALSD) [4] chooses the best of both in time or symbol expansion and is not constrained to search the lattice in a predefined manner like TSD. TSD traversed the search space by making a fixed number of symbol expansions for every frame. ALSD is faster compared to TSD while keeping the same accuracy. Kim et al. [7] further tries to speed up the inference of various optimization such as vectorization of hypotheses and pruning out the redundant search space.

[8] put a very nice study comparing the available beam search decoding strategies for the transducer networks. They also proposed an N-step constrained (NSC) beam search by extending the idea of constraining the beam search to single label emission plus blank label at each time step in [7].

10.21437/Interspeech.2023-2065

3. Recurrent Neural Network Transducers

Transduction loss is an extension of CTC loss for being able to handle any length output sequence. In CTC networks, a label sequence y of length T_y is aligned to an acoustic feature sequence sequence x of length T_x. While CTC networks assume conditional independence, RNN-T networks work by conditioning on the previous tokens generated. Also, RNN-T networks are able to align any label sequence to any other label sequence, while CTC networks only the alignment of sequences, only when $T_y < T_x$.

The idea of RNN-T networks is as follows. Let $x = \{x_1, x_2, ..., x_T\}$ be an input sequence belonging to the set $\mathcal{X}^*$ over an input space $\mathcal{X} = \mathbb{R}^m$, the set of all real valued vectors of dimension m. We want to label this input sequence with an output $y = \{y_1, y_2, ..., y_U\}$ belonging to the set $\mathcal{Y}^*$ over an output space $\mathcal{Y} \subset \mathbb{Z}$, the set of all integer sequences with any length. The objective is to train the function $f : \mathcal{X} \rightarrow \mathcal{Y}$ satisfying all training sample pairs of the style (x, y). The input sequence x is generally some form of short time acoustic feature like filterbanks, MFCCs, etc., while the output sequences are integer tokenized versions of the corresponding transcription. The output space in general is a finite set of integer tokens, the size of the vocabulary used for output text transcriptions of size K. The output space is extended with a *null* token, to $\bar{\mathcal{Y}} = \mathcal{Y} \cup \emptyset$.

The input sequence x is passed through a *transcription* network h^{trans} with the ability to process temporal information, such as a recurrent neural network or a transformer, which is also referred to as the *encoder*. This network produces an output sequence $e \in \mathcal{R}^{T \times D}$ where D is the hidden dimension of the network. The target sequence y is passed though a *prediction* network h^{pred}, also a recurrent or transformer network, also referred to as the *decoder*. This network produces an output sequence $d \in \mathcal{R}^{U \times D}$ using only causal connections. That is

$$d_i = f(y_0, y_1, ..., y_{i-1}) \tag{1}$$

A joiner network h^{join} combines the outputs from the transcription and prediction network producing a 3 dimensional lattice $p \in \mathcal{R}^{T \times U \times K}$. A softmax is taken over the last axis for it to be interpreted as probabilities of vocabulary tokens. $p_{t,u,k}$ indicates the probability of outputting the k^{th} token in the vocabulary in the time-step t, after seeing till the u^{th} token in the sequence. A $\emptyset$ indicates staying in the current token state without generating any new tokens for the current time-step. Let $\mathcal{G}$ be the function which removes $\emptyset$ from a sequence. During training, for a target sequence y, we find all possible sequences $\bar{y} = \mathcal{G}^{-1}(y)$ which can be accommodated in the lattice p, using a forward-backward process and the probability of all paths are maximized.

3.1. Time-synchronous decoding

After training has finished, decoding an input sequence for its most probable sequence can be done in multiple ways. The commonly used scheme is time-synchronous decoding (TSD). In this decoding procedure, we start by providing the input features to the transcription network and a start-of-sequence token $< sos >$ to the prediction network. For a beam search with size N, we start our search procedure at time-step $t = 1$ and token-step $u = 1$, with two sets $A = \{\}$ and $B = \{< sos >\}$. Every candidate will have its own lattice. At every time-step, A is emptied and B is moved to A. If any candidate in A has a proper prefix sequence in A, the probability of the prefix is added to it. The most probable candidate in A removed, u is updated with the number of tokens other than $\emptyset$ and expanded using its most probable token at $p_{t,u}$. If the expanded token is $\emptyset$, that candidate is moved to B, otherwise it is moved to A and the RNN-T lattice for that candidate is expanded with the current new token. When the N^{th} best candidate in B has more probability than the best candidate in A, we move on to the next time-step and update $t := t+1$. The best candidate in A after T time-steps is considered as the most probable output sequence. If we visualize the output lattice of size $T \times U \times K$ with t increasing in the right direction, u in the downwards direction and k in the depth direction, this decode procedure scans from left to right in the lattice from $t = 1$ to $t = T$.

3.2. Alignment-length synchronous decoding

In [4] the authors use an algorithm called Alignment-length synchronous decoding (ALSD) which maintains the condition that all hypotheses in the beam search at every step should have the same number of tokens plus blank/null tokens. This is not to be confused with our approach where the number of non-blank tokens is kept constant at every step. While the emphasis of ALSD is to speed up the decode procedure, our approach merely aims to showcase a decode procedure which is more in line with the training loss. And hence, for all experiments in this paper, we only compare the outputs of our decode scheme with TSD and not ALSD.

4. Prefix-search decoding (PSD)

Drawing heavy inspiration from the prefix-search procedure in CTC-based networks, we formulate a new decoding scheme for decoding sequences from an RNN-T network. Instead of scanning the lattice in a left to right fashion in the $t - axis$, we propose a decode procedure which scans the lattice in a top to down fashion in the $u - axis$. In TSD, the decoding procedure has a practically high probability to stop without any stopping criterion at the T^{th} time-step (that is, the beams don't expand infinitely in a particular time-step). But in our approach, we scan in the direction of the $u - axis$, for which we do not know the length of the sequence U prior to knowing the decoded output. To account for this, we extend the output space with an end-of-sequence token such that $\tilde{\mathcal{Y}} = \bar{\mathcal{Y}} \cup < eos >$. The objective of prefix-search decoding is to consider all possible prefixes of a particular candidate to account for its total probability in the lattice. That is, the probability of a search sequence $y' = \{y_1, y_2, ..., y_u\}$ in the lattice is the total sum of the probability of outputting y' in all time-steps ranging from $t = 0$ to $t = T$, in the lattice. In TSD, the prefixes are considered only if they end up in the best candidates list, while in prefix-search decoding every candidate y' is forced to retain all possible prefixes $\mathcal{G}^{-1}(y')$ ending with y_u of the candidate in the lattice. This makes prefix-search decoding align much closer to the original training objective than TSD, because the training objective is to maximize the probability of all possible paths $\mathcal{G}^{-1}(y)$ of a target sequence y.

Let the first token for any sequence be $< sos >$, and let y be a candidate of length v, and let the input sequence x be of length T. Just like in the original formulation for RNN-T, define the variable $\alpha^{(y)}(t, u)$ as the probability of outputting $y_{[1:u]}$ during time-steps 1to t, and let $Pr(k \in \tilde{\mathcal{Y}}|t, u)$ be the probability of outputting the token k from the vocabulary $\tilde{\mathcal{Y}}$ at time-step t, after only observing the sequence $y_{[1:u]}$ by time-steps 1 to t.

$$\alpha^{(y)}(t,u) = \alpha^{(y)}(t-1,u) \cdot Pr(\emptyset|t-1,u) + \alpha^{(y)}(t,u-1) \cdot Pr(y_u|t,u-1) \quad (2)$$

$$\alpha^{(y)}(t,1) = \begin{cases} 1; & t = 1 \\ 0; & otherwise \end{cases} \quad (3)$$

Then $\gamma^{(y)}(k,t)$, the probability of outputting y during time-steps 1 to t and then outputting the token k from the vocabulary excluding $\emptyset$, can be written as

$$\gamma^{(y)}(k,t) = \alpha^{(y)}(t,v) \cdot Pr(k|t,v) \; \forall k \in \tilde{\mathcal{Y}} \setminus \{\emptyset\} \quad (4)$$

When we encounter an end-of-sequence as the expansion token, all the prefixes have to be joined together such that the paths end at time-step T. Now, the prefix probability of an expanded candidate $y + \{k\}$ can be written as

$$\psi^{(y)}(k) = \begin{cases} \sum_{t=1}^{T} \gamma^{(y)}(k,t) & ; k \notin \{\emptyset, < eos >\} \\ \alpha^{(y+\{<eos>\})}(T, v+1) & ; k =< eos > \\ 0 & ; k = \emptyset \end{cases} \quad (5)$$

Note that for the special cases when the expansion token is $\emptyset$, we set its to probability zero, because null tokens by definition cannot move to the next token-step. In the case where the expansion token is $< eos >$, the score we provide is technically the total probability of that sequence. The pseudo-code for a beam search of size N is given in **Algorithm 1**.

Algorithm 1 Prefix-search decoding

```
ended ← {}
active ← {{< sos >}}
while |ended| < N or Pr(ended[N]) < Pr(active[1]) do
    n ← 1
    while n <= N do
        y ← active[n]
        Pr(y + {k}) = ψ^(y)(k) ∀ k
        best ← N tokens with highest Pr(y + {k}).
        Add (y + {k}) to active ∀k ∈ best ∖ {< eos >}
        if < eos > ∈ best then
            Add (y + {< eos >}) to ended
        end if
        Remove y from active
        n++
    end while
    Sort ended in descending using Pr(y) ∀y ∈ ended
    Sort active in descending using Pr(y) ∀y ∈ active
end while
```

5. Consequences

An unforseen consequence of doing prefix search decoding that our experiments have shown is that PSD requires hyper-parameter tuning in terms of the length penalty that is to be utilized. Unlike TSD, PSD was observed to be more sensitive to changes in length penalties. For a search sequence y, we rescore the candidates every step in the decode as mentioned in [9] and compute the adjusted probability $\tilde{Pr}(y)$ for length penalty λ.

$$log_e(\tilde{Pr}(y)) = log_e(Pr(y)) + \lambda * |y| \quad (6)$$

6. Experimental Setup

6.1. Data Description

Experiments were conducted on the publicly available Librispeech [10] and close-talk AMI datasets [11, 12]. Librispeech data contains the read speech from English audiobooks however AMI dataset has meeting speech recordings. We only use close-talk sets from AMI recorded from individual head microphones (IHM). All the models are trained on TRAIN sets and results are reported on DEV & EVAL sets.

Table 1: *Librispeech dataset details*

Type	Name	#Utt	Hours
TRAIN	train-960	281231	960
DEV	dev-clean	2703	5.4
	dev-other	2864	5.3
TEST	test-clean	2620	5.4
	test-other	2939	5.1

Table 2: *AMI-IHM dataset details*

Type	Name	#Utt	Hours
TRAIN	ihm-train	108221	77.89
DEV	ihm-dev	13059	8.9
TEST	ihm-eval	12612	8.7

6.2. Architecture

We trained an end-to-end ASR system based on Conformer [13] architecture. Our model closely resembles Conformer (S), which is mentioned in [13], having 10.1M parameters. It consists of 16 encoders having conformer blocks and 1 LSTM decoder layer. Encoder and decoder hidden dimensions are 144 & 320 respectively. The kernel size for the convolution module is 32 and attention heads are 4. We use subwords-based output dimensions of 1024 & 256 for Librispeech and AMI datasets respectively. For encoding the text into tokens, we use byte-pair encoding [14].

We use 80-dimension fbank features input using 25ms of window length and with 10ms of hop size. We use Specaugment [15] as a data augmentation strategy.

6.3. Training and inference details

All models were trained with the Noam scheduler [16], with 25000 warm-up steps, a factor of 10, and weight decay of $1e{-}6$. Weight updates are pooled and performed every 24 steps. We decode the model averaged over the last 15 best checkpoints, chosen based on the RNN-T loss criteria calculated over the DEV set. The Pytorch [17] library was used for all trainings and no external language models were used for our experiments. For all experiments, batch binning is used with 2M as batch bin size

and checkpoints are saved every 5000 steps. Models were decoded using 12 as the beam size for all cases. The length penalty for candidate scores were chosen with a grid search from 0.0 to 1.0 with a resolution of 0.1.

7. Results

All our experiments were carried out on the Librispeech corpus and AMI-IHM data. In Table 3, the performance of TSD is compared with that of PSD, with dev-clean, dev-other, test-clean and test-other. Table 4 shows the same experiment carried out on the AMI-IHM dataset, evaluated on the *dev* set and *eval* set.

Table 3: *WER(%) on Librispeech DEV & TEST sets*

Search Type	DEV		TEST	
	clean	other	clean	other
TSD	3.12	3.47	8.43	8.38
PSD (ours)	3.03	3.29	8.18	8.13

Table 4: *WER(%) on AMI-IHM DEV & TEST sets*

Search Type	DEV	TEST
TSD	23.86	23.55
PSD (ours)	24.61	24.13

8. Discussion

With experiments on train-960 set of the Librispeech corpus, we can observe that PSD is always significantly better than TSD. However when we look at the results of AMI-IHM, interestingly the performance of the algorithm dropped significantly. This appears to be a little counter-intuitive. We verified our experiments on a different architecture with libri-960 again, and observe the same trends as in Table 1. In a different experiment not reported in the results for conciseness, we carried out a training with libri-100, and see trends where TSD and PSD perform roughly with the same performance. Our observation is that the discrepancies happen with lower amounts of data such as libri-100 and AMI-IHM. Theory suggests that PSD should always be better than TSD since that is what the loss function tries to minimize, but this is not observed with lower amounts of data. Our hypothesis here is that with lower amounts of data, even though the loss we minimize is for maximizing the probability of all alignments in the lattice, lack of data might be forcing the network to learn only through fewer paths in the lattice, and compounding errors while considering multiple paths that the loss may not have maximized probability over. The clear performance improvements with a larger data size like libri-960 might indicate that with more data, the loss minimization might be able to maximize probabilities through more paths in the lattice. The validation loss values we observe with libri-960 is around 0.10 per token while with AMI-IHM and libri-100 they are around 0.65 and 0.35 respectively, which could be further indication of our hypothesis. There is no clear winner in terms of performance here, but what we can infer is that when dealing with a model trained on sufficient amount of training data, PSD can achieve much better performance when compared to TSD, with the right length penalty.

9. Conclusions

In this work, we introduce a new approach for decoding through the probability lattice of an RNN-T network called prefix-search decoding. We show that this method is closer to the training objective than existing decode strategies. The approach requires no major changes to the existing architecture. We test out this proposed decoding strategy on the Librispeech corpus and observe consistent gains in WER with larger splits such as train-960 having an average relative word error rate gain of 3.5%. Our experiments reveal that even though PSD should in theory be always better than TSD, for low resource scenarios the former has degraded performance in comparison. We conclude that for scenarios where there is an abundance of training data, PSD could outperform TSD.

10. References

[1] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[2] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14. JMLR.org, 2014, p. II–1764–II–1772.

[3] A. Graves, "Sequence transduction with recurrent neural networks," *arXiv preprint arXiv:1211.3711*, 2012.

[4] G. Saon, Z. Tüske, and K. Audhkhasi, "Alignment-length synchronous decoding for rnn transducer," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 7804–7808.

[5] M. Jain, K. Schubert, J. Mahadeokar, C.-F. Yeh, K. Kalgaonkar, A. Sriram, C. Fuegen, and M. L. Seltzer, "Rnn-t for latency controlled asr with improved beam search," *arXiv preprint arXiv:1911.01629*, 2019.

[6] A. Tripathi, H. Lu, H. Sak, and H. Soltau, "Monotonic recurrent neural network transducer and decoding strategies," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2019, pp. 944–948.

[7] J. Kim and Y. Lee, "Accelerating rnn transducer inference via one-step constrained beam search," *arXiv preprint arXiv:2002.03577*, 2020.

[8] F. Boyer, Y. Shinohara, T. Ishii, H. Inaguma, and S. Watanabe, "A study of transducer based end-to-end asr with espnet: Architecture, auxiliary loss and decoding strategies," in *2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2021, pp. 16–23.

[9] S. Kim, T. Hori, and S. Watanabe, "Joint ctc-attention based end-to-end speech recognition using multi-task learning," *Association for Computational Linguistics*, vol. 1, 2017.

[10] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2015, pp. 5206–5210.

[11] J. Carletta, S. Ashby, S. Bourban, M. Flynn, M. Guillemot, T. Hain, J. Kadlec, V. Karaiskos, W. Kraaij, M. Kronenthal *et al.*, "The ami meeting corpus: A pre-announcement," in *Machine Learning for Multimodal Interaction: Second International Workshop, MLMI 2005, Edinburgh, UK, July 11-13, 2005, Revised Selected Papers 2*. Springer, 2006, pp. 28–39.

[12] W. Kraaij, T. Hain, M. Lincoln, and W. Post, "The ami meeting corpus," 2005.

[13] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, "Conformer:

Convolution-augmented Transformer for Speech Recognition," in *Proc. Interspeech 2020*, 2020, pp. 5036–5040.

[14] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *Association for Computational Linguistics*, vol. 1, 2016.

[15] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "Specaugment: A simple data augmentation method for automatic speech recognition," *arXiv preprint arXiv:1904.08779*, 2019.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[17] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.