INTERSPEECH 2019
September 15–19, 2019, Graz, Austria

Vectorized Beam Search for CTC-Attention-based Speech Recognition

Hiroshi Seki[1], *Takaaki Hori*[2], *Shinji Watanabe*[3], *Niko Moritz*[2], *Jonathan Le Roux*[2]

[1] Toyohashi University of Technology, Japan
[2] Mitsubishi Electric Research Laboratories (MERL), USA
[3] Johns Hopkins University, USA

h123346@edu.tut.ac.jp, thori@merl.com, shinjiw@jhu.edu, moritz@merl.com, leroux@merl.com

Abstract

This paper investigates efficient beam search techniques for end-to-end automatic speech recognition (ASR) with attention-based encoder-decoder architecture. We accelerate the decoding process by vectorizing multiple hypotheses during the beam search, where we replace the score accumulation steps for each hypothesis with vector-matrix operations for the vectorized hypotheses. This modification allows us to take advantage of the parallel computing capabilities of multi-core CPUs and GPUs, resulting in significant speedups and also enabling us to process multiple utterances in a batch simultaneously. Moreover, we extend the decoding method to incorporate a recurrent neural network language model (RNNLM) and connectionist temporal classification (CTC) scores, which typically improve ASR accuracy but have not been investigated for the use of such parallelized decoding algorithms. Experiments with LibriSpeech and Corpus of Spontaneous Japanese datasets have demonstrated that the vectorized beam search achieves 1.8$\times$ speedup on a CPU and 33$\times$ speedup on a GPU compared with the original CPU implementation. When using joint CTC/attention decoding with an RNNLM, we also achieved 11$\times$ speedup on a GPU while maintaining the benefits of CTC and RNNLM. With these benefits, we achieved almost real-time processing with a small latency of 0.1$\times$ real-time without streaming search process.

Index Terms: speech recognition, beam search, parallel computing, encoder-decoder network, GPU

1. Introduction

With the success of deep learning [1–5] and the popularization of speech interfaces such as smartphones and smart speakers, there has been unprecedented interest in the development of automatic speech recognition (ASR) systems. In practice, rapid execution of ASR decoding is essential for satisfactory user experience. Reduction of sequence length [5–8] and parallel computing [9–11] have been the main directions investigated to enable rapid computation of likelihoods and efficient traversal of the search space.

Beam search [12] is a breadth-first search algorithm which imposes restrictions on the search space to reduce both memory requirements and computation time. During the search process, hypotheses are expanded from a root node with possible hidden Markov model (HMM) states, phones, or characters. The expanded hypotheses at each depth level (or time-step in the case of time-synchronous beam search for ASR) are stored in a FIFO (first-in, first-out) queue for further expansion at the next depth level. In the beam search, each hypothesis is scored according to some models, and only the top B hypotheses are kept in the queue. Other hypotheses are pruned out and no longer expanded. The number B is called *beam width*. This pruning step plays an important role to reduce the computation time.

In typical ASR decoding, a major part of the computation is dedicated to scoring hypotheses using acoustic and language models such as Gaussian mixture models (GMMs), deep neural networks (DNNs), and recurrent neural networks (RNNs). Thread parallelism and GPU-based execution are effective to accelerate matrix multiplications and element-wise operations required for acoustic score computation. Dixon et al. proposed GPU-based computation of acoustic scores [9]. For further speedup, the search steps for traversing the decoding graph and expanding hypotheses can be parallelized. Chong et al. [10] and Chen et al. [11] extended the search algorithm by executing graph traversal on a GPU. These studies focused on efficient computation of weighted finite-state transducer (WFST) based decoding.

In contrast with these earlier studies, we focus on developing a faster beam search algorithm for attention-based encoder-decoder networks [6, 13]. First, we vectorize the B current hypotheses in the queue and compute posterior probabilities at once for the next time step. This approach can eliminate the for-loop program with regard to the beam width originally managed by the FIFO queue, and exploit parallelism for scoring the hypotheses more effectively. Recently, this type of implementation can be found in Tensorflow [14] for general-purpose beam search decoding[1]. However, such vectorization approach has not been investigated in detail for speech recognition tasks. In this paper, we demonstrate the efficacy of the vectorized beam search using major ASR benchmarks such as LibriSpeech [15] and the corpus of spontaneous Japanese (CSJ) [16].

Second, we extend the beam search to incorporate scores from a recurrent neural network language model (RNNLM) and connectionist temporal classification (CTC) [17–19]. Hybrid approaches that employ attention models, CTC, and RNNLMs have been widely used for end-to-end ASR and shown to improve the recognition accuracy [5, 7, 20, 21], but the effectiveness of parallel search algorithms for such hybrid models is not well investigated. The beam search for the standard attention decoder needs to be performed in an output-label-synchronous manner. A method to combine CTC with the attention decoder uses CTC prefix scores [22], which allows us to compute the CTC scores label-synchronously together with the attention decoder. However, the CTC-based forward probabilities need to be accumulated along the entire sequence of input frames to obtain the prefix scores [23]. This process can be parallelized over multiple hypotheses but not over the input frames. Therefore, if the utterance is long, computation requirements increase, in a quadratic order of the length. To overcome this problem, we propose a frame windowing technique based on the attention weights given by the attention decoder, which can significantly

[1]https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/BeamSearchDecoder

reduce the number of frames considered for computing the prefix scores.

Finally, we further investigate the efficacy of batch processing that decodes multiple utterances simultaneously, where the encoder network generates the hidden vectors of S utterances at once, and the decoder network processes $S \times B$ hypotheses in a mini batch.

2. Beam search

2.1. Definition

Let $\mathcal{H}^l = \{h_1^l, \ldots, h_b^l, \ldots, h_B^l\}$ be a set of hypotheses in the FIFO queue at decoding time step l. Hypothesis h_b^l has its own label history accumulated up to time step l:

$$h_b^l = y_b^1 \cdot y_b^2 \cdots y_b^l, \tag{1}$$

where $y_b^k \in \mathcal{U}$ denotes the k-th output label of h_b^l, and $\mathcal{U} = \{u_1, \ldots, u_U\}$ is an output label set with U elements.

At the next time step $l+1$, for each hypothesis h_b^l, the decoder network generates U new labels with their posterior probabilities, leading to U expanded hypotheses:

$$\bar{h}_{b,i}^{l+1} = h_b^l \cdot u_i, 1 \leq i \leq U. \tag{2}$$

Each hypothesis h_b^l has a score α which is an accumulation of log posterior probabilities up to decoding time step l. The score for the expanded hypotheses can be computed by adding the output of the decoder network:

$$\alpha(\bar{h}_{b,i}^{l+1}) = \alpha(h_b^l) + \log p^{\text{att}}(y_b^{l+1} = u_i | h_b^l, X), \tag{3}$$

where $p^{\text{att}}(y_b^{l+1} | h_b^l, X)$ is a set of posterior probabilities generated by the decoder network and $p^{\text{att}}(y_b^{l+1} = u_i | h_b^l, X)$ the posterior probability for the i-th label u_i. X denotes an input sequence. Following the notations in [24],

$$p^{\text{att}}(y_b^{l+1} | h_b^l, X) = \text{Generate}(\bar{c}_b^{l+1}, r_b^l), \tag{4}$$

$$\bar{r}_{b,i}^{l+1} = \text{Recurrency}(r_b^l, \bar{c}_b^{l+1}, u_i), \tag{5}$$

where r_b^l and $\bar{c}_b^{l+1}$ are the decoder state and the context vector for h_b^l, respectively, and $\bar{r}_{b,i}^{l+1}$ the decoder state for $\bar{h}_{b,i}^{l+1}$. Please refer to [24] for detail.

To reduce the search space, the expanded hypotheses are pruned at each time step. Pruning is performed in a two-step procedure, locally then globally. During local pruning, for each hypothesis h_b^l, the set of U expanded hypotheses $\{\bar{h}_{b,i}^{l+1} | 1 \leq i \leq U\}$ is pruned to its top B hypotheses based on the highest local scores $p^{\text{att}}(y_b^{l+1} = u_i | h_b^l, X)$. Given an array V of scores, we denote by $\text{Top}(V, B)$ the function which returns the B elements in V with highest values, together with their indices in V. Local pruning can then be represented as:

$$\bar{R}_b^{l+1}, \zeta_b^{l+1} = \text{Top}([\log p^{\text{att}}(y_b^{l+1} = u_i | h_b^l, X)]_{1 \leq i \leq U}, B), \tag{6}$$

resulting in a set of B hypotheses (again, for each h_b^l), with scores:

$$\bar{Q}_b^{l+1} = [\alpha(h_b^l) + \log p^{\text{att}}(y_b^{l+1} = u_{\zeta_{b,j}^{l+1}} | h_b^l, X)]_{1 \leq j \leq B}. \tag{7}$$

Local pruning is typically repeated in a for-loop manner to expand all hypothesis of $\mathcal{H}^l$.

During global pruning, the $B \times B$ expanded hypotheses are further pruned down to B hypotheses by picking the hypotheses with B highest scores among all $B \times B$ hypotheses, with indices $\xi^{l+1} = [(b_k^{l+1}, j_k^{l+1})]_{1 \leq k \leq B}$ as follows:

$$Q^{l+1}, \xi^{l+1} = \text{Top}\Big(\bar{Q}^{l+1}, B\Big), \tag{8}$$

$$\mathcal{H}^{l+1} = \{h_b^l \cdot u_{\zeta_{b,j}^{l+1}} | (b, j) \in \xi^{l+1}\}. \tag{9}$$

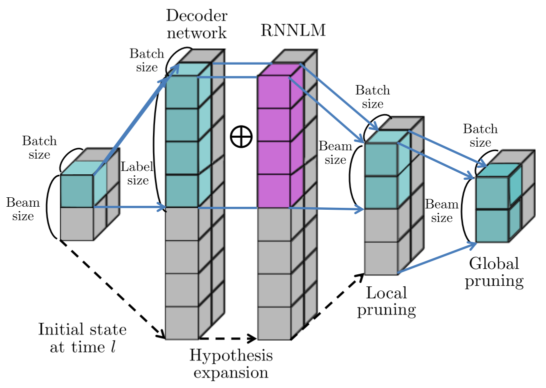

Figure 1: *Procedure of vectorized beam search at time step l. Hypotheses are expanded and added with scores of RNNLM as shallow fusion. The candidate hypotheses are pruned by applying local and global pruning.*

2.2. Implementation

The decoder network in Eq. (4) takes previous label information at step l as input to output the posterior probabilities at step $l+1$. Other than the previous label, the networks with recurrent connections have internal states (e.g., $\bar{r}_{b,i}^{l+1}$ and $\bar{c}_b^{l+1}$ in Eqs. (4) and (5), and the attention weights) which will be used in a future step. These states need to be pruned together with the hypotheses. At the implementation level, each hypothesis is represented as a dictionary data structure including these states and stored in the FIFO queue, allowing pruning to be performed at once for all variables.

3. Vectorized beam search

3.1. Definition

In this section, we reformulate the beam search algorithm of Section 2 by vectorizing the hypotheses and eliminating the loop on the beam size B. We further process S utterances in a batch for reducing computation time in an offline decoding scenario. In online decoding, batch size S is set to 1. Figure 1 shows an overview of the proposed hypotheses expansion and pruning techniques at step l.

For this purpose, we vectorize each element in the dictionary consisting of the internal states as described in Section 2.2. At step $l = 0$, the previous labels are defined as a vector of "start-of-sequence" symbols:

$$\mathbf{y}^0_{[S \times B]} = [\texttt{<sos>}, \ldots, \texttt{<sos>}]^\intercal, \tag{10}$$

and the accumulated scores are defined as:

$$Q^0_{[S \times B]} = [0, \ldots, 0]^\intercal, \tag{11}$$

where the subscript square brackets show the array size, e.g., $Q^0_{[S \times B]}$ is a 1-D array of length $S \times B$. By concatenating S utterances, the encoder network can compute the hidden representations for S utterances at once. The encoder output is then duplicated to B hypotheses to match the number of hypotheses. The decoder network then computes the posterior probabilities for all B beam hypotheses of the S utterances in a batch. Let

$\gamma^{l+1}_{[S\times B,U]}$ be the calculated posterior probabilities for the U expanded hypotheses at step $l+1$ from the $S \times B$ hypotheses at step l. The attention-based decoder network calculations in Eqs. (4) and (5) are replaced by:

$$\gamma^{l+1}_{[S\times B,U]} = \mathrm{Generate}(\bar{\mathbf{c}}^{l+1}_{[S\times B]}, \mathbf{r}^{l}_{[S\times B]}), \quad (12)$$

$$\bar{\mathbf{r}}^{l+1}_{[S\times B,U]} = \mathrm{Recurrency}(\mathbf{r}^{l}_{[S\times B]}, \bar{\mathbf{c}}^{l+1}_{[S\times B]}, \mathbf{u}_{[U]}), \quad (13)$$

where operations are broadcasted appropriately.

After hypothesis expansion, local pruning is applied to reduce the number of hypotheses from U to B for all B hypotheses and S utterances, followed by global pruning to reduce to B hypotheses for each of the S utterances. We define $\mathrm{Top}(V, K, \mathrm{dim})$ as above, with an additional argument indicating the dimension along which selection is performed[2] (as in Python, 0 indicates the first dimension). The accumulated score at time step $l+1$ after local pruning is obtained as:

$$\bar{R}^{l+1}_{[S\times B,B]}, \zeta^{l+1}_{[S\times B,B]} = \mathrm{Top}(\log \gamma^{l+1}_{[S\times B,U]}, B, 1), \quad (14)$$

$$\bar{Q}^{l+1}_{[S\times B,B]} = \alpha^{l}_{[S\times B]} + \bar{R}^{l+1}_{[S\times B,B]}, \quad (15)$$

where the sum is appropriately broadcasted. Here, $\zeta^{l+1}_{(s,b)} = [\zeta^{l+1}_{(s,b),j}]_{1\le j\le B}$ are the indices of the top B output label candidates for hypothesis b of utterance s. The accumulated score $\bar{Q}^{l+1}_{[S\times B,B]}$ is re-sized to $\bar{Q}^{l+1}_{[S,B\times B]}$ for global pruning on the $B \times B$ candidates for each of the S utterances:

$$\bar{Q}^{l+1}_{[S,B\times B]} = \mathrm{Resize}(\bar{Q}^{l+1}_{[S\times B,B]}), \quad (16)$$

$$\bar{\bar{Q}}^{l+1}_{[S,B]}, \xi^{l+1}_{[S,B]} = \mathrm{Top}(\bar{Q}^{l+1}_{[S,B\times B]}, B, 1). \quad (17)$$

Here, $\xi^{l+1}_{s} = [(b^{l+1}_{s,k}, j^{l+1}_{s,k})]_{1\le k\le B}$ is the list of two-dimensional indices of the top B output label candidates within the reshaped $B \times B$ scores for utterance s.

As in Section 2, the other variables are pruned by tracking the selected indices, for example as follows for the hypotheses:

$$\mathcal{H}^{l+1}_{[S\times B]} = \{h^{l}_{(s,b)} \cdot u_{\zeta^{l+1}_{(s,b),j}} | 1 \le s \le S, (b,j) \in \xi^{l+1}_{s}\}. \quad (18)$$

3.2. Shallow fusion of external modules

During beam search, RNNLM and CTC scores can be integrated via shallow fusion. Prior work [5] combines these scores by defining the final log probability $\log p^{\mathrm{hyb}}$ as a weighted sum of CTC prefix score $\log p^{\mathrm{ctc}}$, decoder network score $\log p^{\mathrm{att}}$, and RNNLM score $\log p^{\mathrm{lm}}$:

$$\begin{aligned}\log p^{\mathrm{hyb}}(y^{l+1}_b | h^l_b, X) &= \lambda \log p^{\mathrm{ctc}}(y^{l+1}_b | h^l_b, X) \\ &+ (1-\lambda) \log p^{\mathrm{att}}(y^{l+1}_b | h^l_b, X) \\ &+ \kappa \log p^{\mathrm{lm}}(y^{l+1}_b | h^l_b), \quad (19)\end{aligned}$$

where λ and κ are hyper-parameters controlling each score's contribution. Please refer to [25] for further details. $\log p^{\mathrm{att}}$ in Eq. (3) is replaced by $\log p^{\mathrm{hyb}}$ to include the RNNLM and CTC scores.

3.3. Efficient computation of CTC prefix scores

A CTC prefix score for hypothesis h^l_b is defined as:

$$\log p^{\mathrm{ctc}}(h^l_b, \ldots | X) \triangleq \log \sum_{\nu \in (\mathcal{U} \cup \{\texttt{<eos>}\})^+} p^{\mathrm{ctc}}(h^l_b \cdot \nu | X), \quad (20)$$

where ν represents all possible label sequences except the empty string, and <eos> indicates the end of sentence. Since it is not feasible to consider all possible label sequences, we compute Eq. (20) as:

$$p^{\mathrm{ctc}}(h^l_b, \ldots | X) \propto \sum_t \phi_{t-1}(h^{l-1}_b) p(z_t = h^l_b | X), \quad (21)$$

where $\phi_{t-1}(h^{l-1}_b)$ is the CTC forward probability up to time frame $t-1$ for label sequence h^{l-1}_b. $p(z_t|X)$ denotes the posterior probability distribution at time frame t obtained by a CTC network. In CTC, the probability of all possible future label sequences can be considered a constant independent of h^{l-1}_b. Hence, we ignore the future part of the prefix score.

Eq. (21) can be parallelized over hypothesis index b, but still includes a recursion over time frames, which cannot be parallelized efficiently. This can be a computational bottleneck for long utterances. To avoid this, we propose a windowing technique based on attention weights obtained by the decoder network. Considering that the forward probabilities provide similar input-output alignments to those given by the attention decoder, it is reasonable to limit the frames for the summation around the center frame of attention weights, where the center frame can be computed as an expected value of attended frames using the inner product of the attention weight vector and the frame index vector. We thus limit the range of t in Eq. (21) to $f^l_{\mathrm{start}} \le t < f^l_{\mathrm{end}}$, where

$$f^l_{\mathrm{start}} = \max(\min(f^{l-1}_{\mathrm{end}}, \hat{t}_l - M), 1) \quad (22)$$

$$f^l_{\mathrm{end}} = \max(\hat{t}_l + M, T). \quad (23)$$

$\hat{t}_l$ denotes the center frame at the l-th label, and M is a predefined margin parameter. In Eq. (22), we set the starting frame of the window not to be bigger than the previous end frame f^{l-1}_{end} to avoid disconnecting the recursion of the forward probabilities. In addition, if $\hat{t}_l < \hat{t}_{l-1}$, we set $f^l_{\mathrm{start}} = f^{l-1}_{\mathrm{start}}$ and $f^l_{\mathrm{end}} = f^{l-1}_{\mathrm{end}}$ to maintain the monotonic alignment property.

4. Experiments

4.1. Experimental setup

We used English and Japanese speech corpora, LibriSpeech [15] and CSJ [16]. As input features, we used 80-dimensional log Mel filterbank coefficients and pitch feature with its delta and delta delta features (80+3=83 dimensions) extracted using Kaldi tools [26]. For the LibriSpeech corpus, we used a VGG network followed by a 5-layer BLSTM as the encoder network. The 1st and 2nd pooling layers of VGG network subsampled the hidden vector by a factor of 2 [27]. Each BLSTM layer had 1024 cells in each direction. The decoder network had a 2-layer LSTM with 1024 cells. The number of labels was set to 5,000, consisting of subwords obtained by the SentencePiece method [28]. For the CSJ corpus, we used a VGG network followed by 4-layer BLSTM as the encoder network with the same subsampling technique. Each BLSTM layer had 1024 cells in each direction. The decoder network had a 1-layer LSTM with 1024 cells. The number of labels was set to 3,260 including Japanese Kanji/Hiragana/Katakana characters and special tokens. Beam search decoding was performed using an Intel Core i7-8700K processor with 3.70 GHz for CPU-based experiments and an Nvidia Titan Xp card for GPU-based experiments. In the case of RNNLM shallow fusion and CTC/Attention joint decoding, we used $\lambda = 0.5$ and $\kappa = 0.5$ on LibriSpeech, and $\lambda = 0.3$ and $\kappa = 0.3$ on CSJ. Beam width B was set to 20 in decoding under all conditions. Note that the baseline system for CPU decoding limits the num-

[2]PyTorch supports this function as torch.topk.

Table 1: *Baseline performance on LibriSpeech.*

	test-clean		test-other	
	%WER	RTF	%WER	RTF
ATT	5.7	1.13	17.2	1.12
+RNNLM	5.2	1.14	16.2	1.14
+CTC	4.6	1.24	13.7	1.26

Table 2: *Baseline performance on CSJ.*

	eval-1		eval-2		eval-3	
	%CER	RTF	%CER	RTF	%CER	RTF
ATT	8.5	1.28	6.1	1.29	7.0	1.26
+RNNLM	7.9	1.30	5.8	1.31	6.7	1.28
+CTC	7.3	1.36	5.2	1.40	6.2	1.34

Table 3: *Effect of vectorization on speed for LibriSpeech. Speed is measured in RTF on the test-clean set.*

	ATT	+RNNLM	+CTC
Baseline (CPU)	1.13	1.14	1.24
Vectorized (CPU)	0.62	0.64	0.71
Vectorized (GPU)	0.03	0.03	0.51

Table 4: *Effect of vectorization on speed for CSJ. Speed is measured in RTF on the eval-1 test set.*

	ATT	+RNNLM	+CTC
Baseline (CPU)	1.28	1.30	1.36
Vectorized (CPU)	0.68	0.70	0.74
Vectorized (GPU)	0.04	0.04	0.62

ber of labels scored by CTC to $B \times 1.5$ for efficient computation, where the labels are selected using the attention model scores. The same technique was also used in the vectorized beam search for CPU decoding, but not for GPU decoding. We trained RNNLMs using the transcriptions of LibriSpeech and CSJ. Each model had a 2-layer LSTM with 650 cells. We used the end-to-end speech processing toolkit ESPnet [25] for training ASR models and testing decoding algorithms.

4.2. Baseline performance

Tables 1 and 2 show baseline ASR performance for the LibriSpeech and CSJ tasks. Recognition accuracy is represented as word error rate (%WER) for LibriSpeech test sets (test-clean and test-other) and character error rate (%CER) for CSJ test sets (eval-1, eval-2, and eval-3). The decoding speed is measured in real-time factor (RTF), which is the ratio of decoding time to the utterance length, and therefore a smaller RTF indicates faster decoding. We tested different decoder conditions including attention decoder only (ATT), attention decoder with RNNLM (+RNNLM), and CTC/attention joint decoding with RNNLM (+CTC). The recognition errors were reduced by adding an RNNLM and CTC, but the decoding time increased with these components. Note that the baseline decoder in ESPnet is written in pure Python code and not optimized in terms of speed. The speed is thus insufficient for real applications.

4.3. Performance of vectorized beam search

We measured the decoding speed when using the vectorized beam search. Tables 3 and 4 show the RTFs with and without vectorization for the LibriSpeech and CSJ tasks. We evaluated the speed both on CPU and GPU. Note that since there is no accuracy degradation by vectorization, we do not show WER or CER here. As can be seen, the vectorized beam search provided significant speed improvement from the baseline on both CPU and GPU. In the case of the attention decoder with RNNLM, we obtained 1.8× speedup on CPU and 33× speedup on GPU. In joint decoding with CTC, the vectorization increased the speed by 1.7 and 2.4 on CPU and GPU, respectively. However, the improvement in GPU decoding was relatively small unlike the case of attention decoder with RNNLM, since there is a bottleneck in CTC score computation as described in Section 3.3.

Table 5: *Effect of frame windowing for CTC prefix scores. Speed is measured in RTF on the LibriSpeech test-clean and CSJ eval-1 test sets.*

Margin parameter in Eqs. (22) and (23)	LibriSpeech		CSJ	
	%WER	RTF	%CER	RTF
$M = \infty$	4.6	0.51	7.3	0.62
$M = 60$	4.6	0.14	7.3	0.17
$M = 50$	4.6	0.11	7.3	0.14
$M = 40$	**4.6**	**0.09**	7.3	0.12
$M = 30$	4.9	0.08	**7.3**	**0.11**
$M = 20$	5.2	0.07	7.6	0.09

Table 6: *Speed improvement by batch processing. Speed is measured in RTF on the LibriSpeech test-clean and CSJ eval-1 test sets. Frame windowing was applied with $M = 40$ for CTC.*

Batch size	LibriSpeech		CSJ	
	+RNNLM	+CTC	+RNNLM	+CTC
$S = 1$	0.03	0.09	0.04	0.12
$S = 2$	0.02	0.06	0.03	0.08
$S = 4$	0.02	0.05	0.03	0.06
$S = 8$	0.02	0.04	0.02	0.05

4.4. Frame windowing for CTC prefix scores

To reduce the CTC score computation, we implemented the frame windowing technique proposed in Section 3.3. Table 5 shows the relationship between the error rate and the decoding time when changing margin parameter M. In both LibriSpeech and CSJ, even when reducing M to 40, the error rate did not increase at all while achieving almost 5× faster decoding. These results demonstrate that frame windowing is quite effective for computing CTC prefix scores in joint decoding.

4.5. Evaluation of batch decoding

The vectorized beam search can be applied not only for single utterances but also multiple utterances in a batch. We evaluated the decoding speed when we increased the batch size S. Table 6 shows RTFs for different values of S. We obtained RTF reductions as the batch size increased, which demonstrates that the algorithm successfully works also in batch processing. However, the reduction ratios were relatively small for the larger batch sizes. This could be due to some overhead for GPU memory allocation during the beam search since the batch processing requires an $S\times$ larger space. We will consider this problem in future work.

5. Conclusions

In this paper, we investigated a vectorized beam search technique to speed up attention-based end-to-end speech recognition. By vectorizing hypotheses, we achieved a speedup compared with the original beam search algorithm of 1.85× on the LibriSpeech corpus, and 1.79 × on the CSJ corpus. In the case of GPU-based execution, we further achieved 18.8× speedup on LibriSpeech and 17.5× speedup on CSJ. For CTC/attention joint decoding, we proposed a frame windowing technique, which reduced the decoding time by a factor of 5 on GPU. We also examined the batch processing of multiple utterances, and confirmed further speedups as the batch size increased.

6. References

[1] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig, "The Microsoft 2016 conversational speech recognition system," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2017, pp. 5934–5938.

[2] K. Audhkhasi, B. Kingsbury, B. Ramabhadran, G. Saon, and M. Picheny, "Building competitive direct acoustics-to-word models for english conversational speech recognition," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Apr. 2018, pp. 4759–4763.

[3] C.-C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, N. Jaitly, B. Li, J. Chorowski, and M. Bacchiani, "State-of-the-art speech recognition with sequence-to-sequence models," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Apr. 2018, pp. 4774–4778.

[4] J. Li, G. Ye, A. Das, R. Zhao, and Y. Gong, "Advancing acoustic-to-word CTC model," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Apr. 2018, pp. 5794–5798.

[5] T. Hori, S. Watanabe, Y. Zhang, and C. William, "Advances in joint CTC-Attention based end-to-end speech recognition with a deep CNN encoder and RNN-LM," in *Proc. Interspeech*, Aug. 2017, pp. 949–953.

[6] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2016, pp. 4960–4964.

[7] A. Zeyer, K. Irie, R. Schlüter, and H. Ney, "Improved training of end-to-end attention models for speech recognition," in *Proc. Interspeech*, Sep. 2018, pp. 7–11.

[8] N. Moritz, T. Hori, and J. Le Roux, "Triggered attention for end-to-end speech recognition," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019.

[9] P. R. Dixon, T. Oonishi, and S. Furui, "Harnessing graphics processors for the fast computation of acoustic likelihoods in speech recognition," *Computer Speech & Language*, vol. 23, no. 4, pp. 510–526, 2009.

[10] J. Chong, E. Gonina, Y. Yi, and K. Keutzer, "A fully data parallel WFST-based large vocabulary continuous speech recognition on a graphics processing unit," in *Proc. Interspeech*, Sep. 2009, pp. 1183–1186.

[11] Z. Chen, J. Luitjens, H. Xu, Y. Wang, D. Povey, and S. Khudanpur, "A GPU-based WFST decoder with exact lattice generation," in *Proc. Interspeech*, Sep. 2018, pp. 2212–2216.

[12] X. L. Aubert, "An overview of decoding techniques for large vocabulary continuous speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 89–114, 2002.

[13] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio, "End-to-end attention-based large vocabulary speech recognition," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2016, pp. 4945–4949.

[14] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proc. USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Nov. 2016, pp. 265–283.

[15] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "LIBRISPEECH: An ASR corpus based on public domain audio books," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Apr. 2015, pp. 5206–5210.

[16] K. Maekawa, "Corpus of Spontaneous Japanese: Its design and evaluation," in *Proc. ISCA & IEEE Workshop on Spontaneous Speech Processing and Recognition*, Apr. 2003.

[17] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *Proc. International Conference on Machine learning (ICML)*, Jun. 2006, pp. 369–376.

[18] Y. Miao, M. Gowayyed, and F. Metze, "EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding," in *Proc. IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, Dec. 2015, pp. 167–174.

[19] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in English and Mandarin," in *Proc. International Conference on Machine Learning (ICML)*, Jun. 2016, pp. 173–182.

[20] A. Das, J. Li, R. Zhao, and Y. Gong, "Advancing connectionist temporal classification with attention modeling," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Apr. 2018, pp. 4769–4773.

[21] A. Kannan, Y. Wu, P. Nguyen, T. N. Sainath, Z. Chen, and R. Prabhavalkar, "An analysis of incorporating an external language model into a sequence-to-sequence model," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Apr. 2018, pp. 1–5828.

[22] T. Hori, S. Watanabe, and J. R. Hershey, "Joint CTC/attention decoding for end-to-end speech recognition," in *Proc. Annual Meeting of the Association for Computational Linguistics (ACL)*, Jul. 2017.

[23] A. Graves, "Supervised sequence labelling with recurrent neural networks," *PhD thesis, Technische Universität München*, 2008.

[24] S. Kim, T. Hori, and S. Watanabe, "Joint CTC-attention based end-to-end speech recognition using multi-task learning," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2017, pp. 4835–4839.

[25] S. Watanabe, T. Hori, S. Karita, T. Hayashi, J. Nishitoba, Y. Unno, N.-E. Y. Soplin, J. Heymann, M. Wiesner, N. Chen *et al.*, "ESPnet: End-to-end speech processing toolkit," in *Proc. Interspeech*, Sep. 2018, pp. 2207–2211.

[26] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The Kaldi speech recognition toolkit," in *Proc. IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, Dec. 2011.

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[28] T. Kudo and J. Richardson, "SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," in *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Oct. 2018, pp. 66–71.