ACCELERATING RECURRENT NEURAL NETWORK LANGUAGE MODEL BASED ONLINE SPEECH RECOGNITION SYSTEM

Kyungmin Lee, Chiyoun Park, Namhoon Kim, and Jaewon Lee

DMC R&D Center, Samsung Electronics, Seoul, Korea

{k.m.lee, chiyoun.park, namhoon.kim, jwonlee}@samsung.com

ABSTRACT

This paper presents methods to accelerate recurrent neural network based language models (RNNLMs) for online speech recognition systems. Firstly, a lossy compression of the past hidden layer outputs (history vector) with caching is introduced in order to reduce the number of LM queries. Next, RNNLM computations are deployed in a CPU-GPU hybrid manner, which computes each layer of the model on a more advantageous platform. The added overhead by data exchanges between CPU and GPU is compensated through a frame-wise batching strategy. The performance of the proposed methods evaluated on LibriSpeech[1] test sets indicates that the reduction in history vector precision improves the average recognition speed by 1.23 times with minimum degradation in accuracy. On the other hand, the CPU-GPU hybrid parallelization enables RNNLM based real-time recognition with a four times improvement in speed.

Index Terms— Online speech recognition, language model, recurrent neural network, graphic processing unit

1. INTRODUCTION

A language model (LM) computes the likelihood of a given sentence and is used to improve the accuracy of an automatic speech recognition (ASR) system. Recent research has focused on neural network (NN) based LMs [1] because of their outstanding performances in generalizing from sparse data, which traditional n-gram based LMs could not do. In particular, recurrent neural network based LMs (RNNLMs) [2] do not even require Markov assumptions as they can model word histories of variable-length, and these virtues of them have helped improve the performance of many ASR systems [3, 4]. However, to our knowledge, they are not yet actively adopted in real-time ASR systems due to their high computational complexities.

Several attempts have been made to utilize RNNLMs for online decoding in real-time ASR systems [5, 6, 7] However, they either simulate only some aspects of RNNLMs into the traditional architectures [5, 6], or perform a 2-pass decoding [7] which innately could not be applied before the end of the utterance was reached. There have also been attempts to apply RNNLM directly to online ASR without approximation by eliminating redundant computations [8, 9, 10]. In our previous research [9], we were successful in applying moderate size RNNLMs directly to CPU-GPU hybrid online ASR systems with a cache strategy [10]. However, in order to apply it to a more complex task with bigger RNNLMs, we needed to find a way to accelerate it further.

Recent studies indicate that one can reduce the number of distinct RNN computations by treating similar past hidden layer outputs, also referred to as history vectors, as same [11], and that RNNLMs can be accelerated with GPU parallelization [12]. In this paper, we attempt two different approaches in order to achieve real-time performance in a large RNNLM based ASR system. Firstly, a lossy compression is applied to the cache of the history vector. The precision of the vectors can be controlled by either rounding up with a smaller number of significant digits or at an extreme, by storing only the sign of each element. Next, we propose GPU parallelization of RNNLM computations, but only on selected layers. Instead of performing all RNNLM computations on the same platform, compute-intensive parts of the model are computed on GPUs, and the parts that need to utilize a large memory are calculated on CPUs. This method inherently increases the overhead of data transfer between CPUs and GPUs. This is handled by coordinating a batch transfer method that reduces the number of communications and the size of the data blocks at the same time in the hybrid ASR systems.

The paper is organized as follows. The architecture of our baseline ASR system is explained in Section 2. The lossy compression method of the history vectors is explained in Section 3. Section 4 explains how RNNLM rescoring is accelerated with CPU-GPU parallelization. Section 5 evaluates performance improvements of the proposed methods, followed by conclusion in Section 6.

2. ARCHITECTURE OF OUR BASELINE CPU-GPU HYBRID RNNLM RESCORING

In the CPU-GPU hybrid ASR system [13], the weighted finite state transducer (WFST) is composed of four layers each representing an acoustic model (AM), a context model, a pronunciation model, and an LM. WFSTs output word hypotheses when they reach word boundaries during frame-synchronous Viterbi searches and the hypotheses can be rescored by a separately stored RNNLM. However, in order to speed up on-the-fly rescoring based on RNNLMs, we needed to reduce redundant computations as much as possible. In this section, we briefly outline the architecture of our baseline CPU-GPU hybrid RNNLM rescoring proposed in [9]. The main highlights of our baseline architecture are the use of gated recurrent unit (GRU) [14] based RNNLM, noise contrastive estimation (NCE) [15] at the output layer, n-gram based maximum entropy (MaxEnt) bypass [16] from input to output layers, and cache based on-the-fly rescoring.

2.1. GRU based RNN

We employed a GRU which is a type of gated RNNs [14]. The GRU is a mechanism designed to prevent vanishing gradient problems related to long-term dependencies over time by using reset gates and update gates. For calculating a output vector of a GRU hidden layer, a total of six weight matrices and three bias vectors need to be loaded into memory since for each gate and a candidate activation, two

[1] http://www.openslr.org/12/

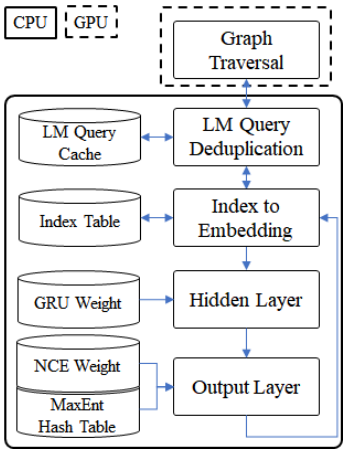

Fig. 1. On-the-fly rescoring with the LM query cache (baseline).

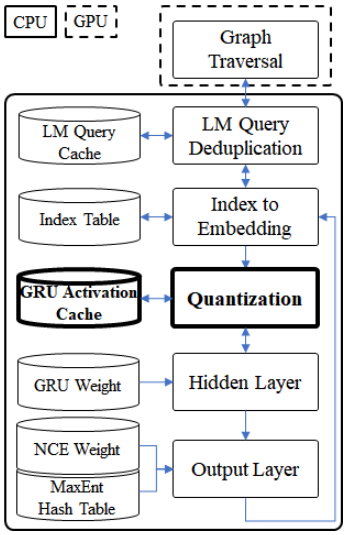

Fig. 2. Proposed on-the-fly rescoring with the cache of quantized history vectors.

weight matrices and one bias vector are required. Thus the memory usage can go up to several megabytes even if the weights are stored in a single precision floating-point format. The computational complexities of GRU computations are $O(H \times H)$ for a hidden layer of size H. This is a highly compute-intensive task considering that the number of unique LM queries in decoding an utterance can reach several hundreds of thousands.

2.2. Noise contrastive estimation

In order to guarantee that the scores calculated at the output layer of an RNNLM are valid probabilities, they need to be normalized over different word sequences. The normalization is a highly computationally intensive task considering the vocabulary size V can reach millions. In order to address this, we employ an NCE at the output layer [15]. NCE is a sampling-based approximation method that treats partition functions as separate parameters and learns them by non-linear logistic regression. The variances of these partition functions estimated by NCE are often limited to small values [17], allowing us to use the unnormalized scores without significant reduction in the recognition accuracy. Even though the only required computations are inner products between the GRU outputs and NCE weights corresponding to the current word, the NCE weight matrix of size $H \times V$ need to be loaded into memory.

2.3. Maximum Entropy

The second strategy to reduce computation in our GRU based RNNLM is to use an n-gram based MaxEnt bypass connections from input to output layers [16]. The MaxEnt scheme helps in maintaining a relatively small size for the hidden layer without significant reduction in recognition accuracy. The two types of parallel models, the main network consisting of GRUs and NCE, and the other with MaxEnt bypass connections, operate as an ensemble model and can improve the overall recognition accuracy. In order to reduce the computational overhead because of the bypass connections, we implemented a hash-based MaxEnt. This method requires the loading of a large hash table proportional to the number of n-grams, to retrieve a probability for the given n-gram in constant time.

2.4. On-the-fly rescoring with cache

The process flow diagram of our baseline CPU-GPU hybrid RNNLM rescoring is shown in Figure 1. The LM queries with same history as well as following words are deduplicated by applying a cache strategy at the start of the rescoring procedure [9]. After the deduplication, the embedding vectors corresponding to indices are retrieved by using an "Index Table". The RNNLM computations are then performed with appropriate values in CPU memory. The results of the calculations are converted to indices, cached, and returned to graph traversals.

3. QUANTIZATION OF HISTORY VECTORS

The cache-based strategy for deduplicating LM queries dramatically accelerated our baseline RNNLM rescoring with a cache hit ratio of around 89% and more than 10 times reduction in computation time [9]. However, there is still room for improvement by extending this caching strategy to the outputs of GRU hidden layers.

The current GRU hidden layer outputs computed based on the previous GRU hidden layer outputs (history vectors) could be shared between similar LM queries. Therefore, in order to reuse the precomputed history vectors, we created another cache for that vectors just before computing RNNLMs as shown in Figure 2. The key of the cache is the GRU input which is a pair of a word embedding and a history vector, and the value of the cache is a GRU hidden layer output corresponding to that input. The number of unique computations is further reduced by assuming that close history vectors would result in similar GRU hidden layer outputs, with negligible effect on the overall ASR results. Euclidean distance would be an easy way to measure the similarity [11], but it would still require a lot of computations that can slow down the whole rescoring process. Instead, we propose to quantize the history vectors by controlling the precision of history vector itself by rounding up to a specified decimal point. We also consider an extreme case, in which we store only the signs of each element, as it would still capture some of the latent meanings which the hidden layers represent.

Table 1 shows the possible reduction of computations for a

Table 1. Redundancy rates of quantized history vectors.

Precision	Count	Redundancy rate
(baseline)	103,904	0.0 %
round-2	102,776	1.09 %
round-1	102,776	1.09 %
sign	88,749	14.59 %

four-second utterance. (Note that each element of the history vector ranges from -1 to 1.) The term "Precision" refers to the quantization of history vectors to a specified decimal place. After the initial deduplication, in our baseline system, we have 103,904 unique LM queries as can be observed from the first row of Table 1. The "round-2" row shows that only 1.09% of the computations can be reduced by caching the history vectors rounded to the second decimal place. Rounding off the history vectors to the first decimal place shows that there is no further redundancy. However, as shown in the last row of Table 1, an extreme case of quantization where only signs of each element are stored, we were able to reduce 14.59% of the computations. This relatively huge reduction may affect the accuracy of RNNLM results to some extent since after the extreme sign quantization there are still 2^{256} possible unique history vectors for a hidden layer of size $H = 256$, but it is worthy to evaluate its effect on ASR systems.

4. CPU-GPU HYBRID DEPLOYMENT OF RNNLM COMPUTATION

As described in Section 2, the proposed RNNLM model cannot be readily deployed on a GPU processor due to its large memory requirement. The word embedding step at the input layer requires space proportional to the size of vocabulary, and the MaxEnt step at the output layer need to maintain a large hash table that can store the n-grams and the corresponding scores. Also, the NCE step at the output layer requires loading of an NCE weight matrix proportional to the size of vocabulary. On the other hand, the hidden layer occupies only a fixed amount of memory but needs a large number of computations instead.

Table 2. Operation times for each RNNLM computation step in seconds.

Processor	Data transfer			Hidden Layer	Output Layer
	Unit	Count	Time		
CPU	-	-	-	6.23	0.04
GPU	LM Query	102,172	5.94	2.15	0.06
GPU	Frame	518	0.60	2.26	0.03

The first row of Table 2 shows a profiling result of an RNNLM computation with a single layer of 128 GRU nodes based on a three-second utterance. As is expected, the hidden layer takes 99% of the overall computation, which we aim to reduce in this section. The high computational rates of neural networks are easily accelerated by utilization of GPUs, but their high memory requirements for word embeddings and MaxEnts prevent us from doing so. Therefore, we deploy only the hidden layer part of the computation on the GPUs and keep the input embedding and output layer computations on the CPU side, as shown in Figure 3. As can be observed from the second row of Table 2, the hybrid deployment reduces the computation time for the hidden layer to one-third of what was done on CPU alone.

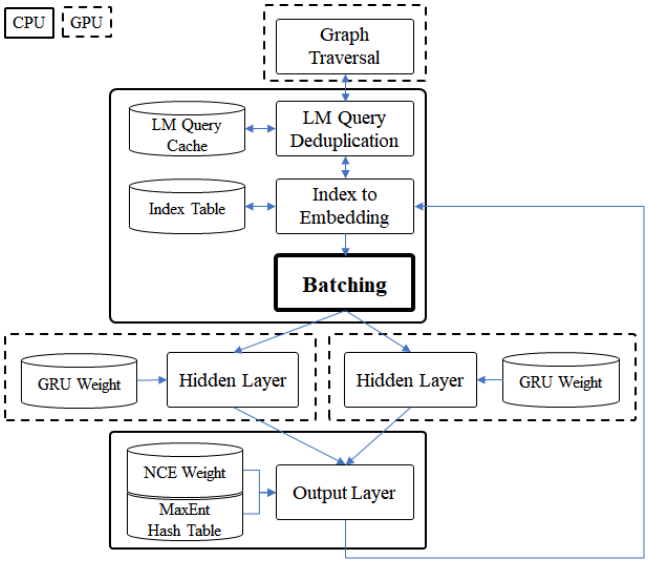

Fig. 3. Proposed GPU based RNNLM rescoring with frame-wise batch data transfer.

However, this method also introduces a setback. Because only the middle layer of the RNNLM computations was deployed on the GPU side, and its surrounding layers are computed on CPUs, the information needs to be shared across the two heterogeneous processor units frequently. As the number of data exchanges increases, the decoding speed of the hybrid ASR system inevitably decreases. The second row of Table 2 also shows that there have been more than a hundred thousand data exchanges during an utterance, which delayed the overall computation by 5.94 seconds, which is twice as long as the original utterance.

The frequency of data transfers between CPUs and GPUs affects the decoding speed more critically than the data size in each transfer. Therefore, we propose a method in which we reduce the number of data copies between CPUs and GPUs by concatenating the needed information to one block per frame. During the batching step, the history vectors and their next word embeddings that are emitted for each frame are stored in a consecutive CPU memory block, and the whole data block is transferred to GPU memory at once. The GRU outputs from the GPU are also copied back to the output layer computation in one data block. This effect can be observed from the last row of Table 2, in which the data transfer time is reduced to 10% of the original. In addition, this approach still works in multi-GPU environments without additional operations by evenly distributing the block to GPUs since the hidden layer calculations for each segment of the CPU memory block are not sequentially related to each other.

5. EXPERIMENTS

5.1. Experimental setup

The LMs in our experiments were trained on the training corpus of LibriSpeech [18]. To compare the performance with n-grams, "4-gram full LM" in LibriSpeech was used. Both vanilla-RNNLMs and GRU-RNNLMs consisted of a single hidden layer and 4-gram based MaxEnt connections. The vocabularies used for all RNNLMs were the same as "4-gram full LM" ($V = 200,000$). A bi-directional recurrent deep neural network (RDNN) based AM with three hidden

Table 3. Performances on LibriSpeech's test sets; all evaluations were performed with same decoding options.

LM	Processor	Rescoring threads	Precision	dev-clean		test-clean		dev-other		test-other	
				WER	RTF	WER	RTF	WER	RTF	WER	RTF
4-gram full	CPU	4	-	4.28	0.18	4.95	0.33	11.92	0.54	11.87	0.26
GRU-RNNLM ($H = 256$)	CPU	4	(baseline)	4.05	2.16	4.69	2.19	11.70	3.58	11.47	3.37
			round-2	4.06	1.85	4.69	1.89	11.69	2.91	11.49	2.85
			round-1	4.05	1.82	4.69	1.87	11.69	2.95	11.48	2.89
			sign	4.06	1.79	4.69	1.80	11.69	2.82	11.47	2.75
	GPU	1	-	4.05	1.10	4.69	1.08	11.70	1.94	11.49	2.29
		2		4.06	0.71	4.69	0.70	11.70	1.24	11.47	1.20
		3		4.05	0.58	4.68	0.63	11.69	0.98	11.47	0.97
		4		4.05	0.52	4.69	0.52	11.70	0.88	11.47	0.94

long short term memory (LSTM) layers (500 nodes for each layer), and a softmax output layer was trained using about 7,600 hours of the fully transcribed in-house English speech data mostly consisting of voice commands and dialogs. WFSTs were compiled with 2-gram LMs, and all the epsilon transitions were removed so that computations on GPUs could be optimized.

The hardware specification for the evaluations was Intel Xeon E5-2680 with 12 physical CPU cores and four Nvidia Tesla K80 GPUs equipped with 12 GB memory. We used CUDA for GPU parallelization. CUBLAS, which is a linear algebra library of CUDA, was used for matrix multiplications and kernel functions were implemented for relatively simple operations such as element-wise operations. For RNNLM computations on CPUs such as output layer computations, we used EIGEN which is a C++ based linear algebra library.

5.2. Results

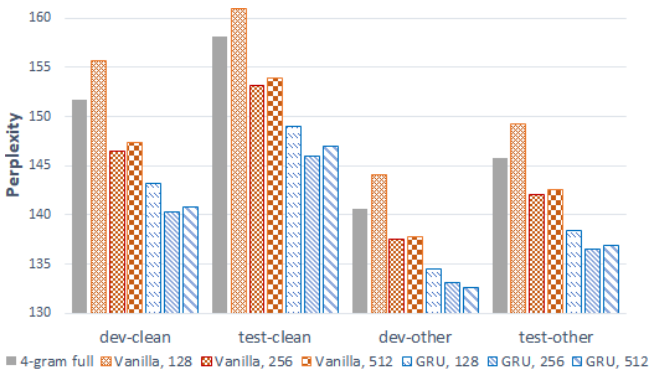

Fig. 4. Perplexities depending on LM types.

In our experiments, LibriSpeech's development and test cases were used for evaluations. The performance of different LMs measured in terms of perplexity is shown in Figure 4. The term "other" in the evaluation cases means the speech data sets were recorded in noisy environments. As can be seen in Figure 4, the vanilla-RNNLM of size 128 showed the worst accuracies over all the sets and was even worse than that of the 4-gram LM. The accuracy of vanilla-RNNLM improved dramatically for a hidden layer size of 256 and showed the lowest perplexities, but still worse than a 128-size GRU-RNNLM. Perplexities of GRU-RNNLMs were dropped by 7.81, 10.10, and 9.75 absolute (averaged over all four cases) for model sizes of 128, 256, and 512, respectively, as compared to the perplexity of the 4-gram LM. In all tasks except for "dev-other," the GRU-RNNLM size of 256 showed the lowest LM perplexities.

Table 3 shows the word error rate (WER) and the real-time factor (RTF) for the proposed methods for accelerating the online RNNLM rescoring. All decoding options otherwise mentioned in Table 3 are same for all the methods being compared. The meanings of values in the column "Precision" are the same as Table 1. Regarding recognition accuracies, the average WER of the baseline system was improved by 3.39% relatively than that of the 4-gram LM based system as can be observed from the first two rows of Table 3. As expected in Section 3, caching quantized history vectors rounded off in the first and the second decimal points did not show noticeable improvement in recognition speed compared to the baseline system. However, the proposed quantization strategy of caching only signs of the history vectors was 1.23 times faster compared to the baseline system without any accuracy degradations.

As shown in the fifth and sixth rows of Table 3, with the proposed GPU parallelization method, even one thread was 1.43 times faster on an average than the fastest CPU based system (sign). The recognition speed improves further with the use of multiple GPUs. In particular, when the number of GPUs increased to two, the speed was significantly improved, which was 1.61 times faster than a single GPU-based system. When three GPUs were utilized, we attained real-time speech recognition over all the test cases. Finally, the RNNLM-based ASR system with four GPUs has shown the fastest average recognition speed of 0.72 RTF over all four test cases. It was three times faster than the fastest CPU-based system and four times faster than the baseline system.

6. CONCLUSION

We devised a faster RNNLM based on-the-fly rescoring on both CPU and GPU platforms by introducing a lossy compression strategy of history vectors and the novel hybrid parallelization method. As cache hit ratios got higher by lowering decimal precisions of the vectors, speech recognition was speeded up by 1.23 times. Although it was not a significant improvement, the fact that recognition rates were not affected even if each dimension of the history vectors was stored by one bit representing the sign seemed to provide a clue to the efficient compression way of embedding vectors while minimizing the loss of their information. Finally, with the CPU-GPU hybrid parallelization method, the decoding speed over all the cases has fallen within real-time.

7. REFERENCES

[1] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[2] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur, "Recurrent neural network based language model," in *Proc. Interspeech*, 2010, pp. 1045–1048.

[3] S. Kombrink, T. Mikolov, M. Karafiat, and L. Burget, "Recurrent neural network based language modeling in meeting recognition," in *Proc. Interspeech*, 2011, pp. 5528–5531.

[4] O. Tilk and T. Alumäe, "Multi-domain recurrent neural network language model for medical speech recognition," in *Proc. Human Language Technologies*, 2014, vol. 268, pp. 149–152.

[5] L. Gwnol and M. Petr, "Conversion of recurrent neural network language models to weighted finite state transducers for automatic speech recognition," in *Proc. Interspeech*, 2012, pp. 131–134.

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

[7] Y. Si, Q. Zhang, T. Li, J. Pan, and Y. Yan, "Prefix tree based n-best list re-scoring for recurrent neural network language model used in speech recognition system," in *Proc. Interspeech*, 2013, pp. 3419–3423.

[8] T. Hori, Y. Kubo, and A. Nakamura, "Real-time one-pass decoding with recurrent neural network language model for speech recognition," in *Proc. ICASSP*, 2014, pp. 6364–6368.

[9] K. Lee, C. Park, I. Kim, N. Kim, and J. Lee, "Applying gpgpu to recurrent neural network language model based fast network search in the real-time lvcsr," in *Proc. Interspeech*, 2015, pp. 2102–2106.

[10] Z. Huang, G. Zweig, and B. Dumoulin, "Cache based recurrent neural network language model inference for first pass speech recognition," in *Proc. ICASSP*, 2014, pp. 6354–6358.

[11] X. Liu, X. Chen, Y. Wang, M. Gales, and P. Woodland, "Two efficient lattice rescoring methods using recurrent neural network language models," *IEEE/ACM Trans. Audio, Speech & Language Processing*, vol. 24, no. 8, pp. 1438–1449, 2016.

[12] X. Chen, Y. Wang, X. Liu, M. Gales, and P. Woodland, "Efficient gpu-based training of recurrent neural network language models using spliced sentence bunch," in *Proc. INTERSPEECH*, 2014, pp. 641–645.

[13] J. Kim, J. Chong, and I. Lane, "Efficient on-the-fly hypothesis rescoring in a hybrid gpu/cpu-based large vocabulary continuous speech recognition engine," in *Proc. INTERSPEECH*, 2012, pp. 1035–1038.

[14] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014.

[15] M. Gutmann and A. Hyvärinen, "Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics," *Journal of Machine Learning Research*, vol. 13, pp. 307–361, 2012.

[16] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Cernocký, "Strategies for training large scale neural network language models," in *Proc. ASRU*, 2011, pp. 196–201.

[17] X. Chen, X. Liu, M. Gales, and P. Woodland, "Recurrent neural network language model training with noise contrastive estimation for speech recognition," in *Proc. ICASSP*, 2015, pp. 5411–5415.

[18] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *Proc. ICASSP*, 2015, pp. 5206–5210.