ACCELERATING RECURRENT NEURAL NETWORK LANGUAGE MODEL BASED ONLINE SPEECH RECOGNITION SYSTEM

Authors: Kyungmin Lee, Chiyoun Park, Namhoon Kim, and Jaewon Lee / **Affiliation**: SR Research(former DMC R&D Center), Samsung Electronics, Seoul, Korea

SAMSUNG Research

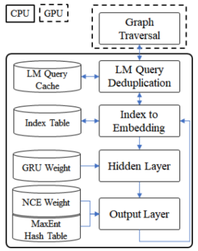
Motivation & Problems

- **RNNLMs** do not 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.
- **However**, to our knowledge, they are not yet actively adopted in real-time ASR systems due to their high computational complexities.

Architecture of Recurrent Neural Network

- **GRU based RNN – Compute intensive task**
  - ▲ **Memory** usage can go up to several megabytes in practice.
  - ▼ The computational **complexities of GRU computations** are $O(H \times H)$ for a hidden layer of size H.
- **Noise contrastive estimation – Memory intensive task**
  - ▲ The required **computations** are inner products between the GRU outputs and NCE weights corresponding to the current word.
  - ▼ NCE weight matrix of size $H \times V$ needs to be loaded into **memory for vocabularies** of size V.
- **Maximum Entropy – Memory intensive task**
  - ▲ We employed a **hash-based** MaxEnt.
  - ▼ This method requires the loading of **a large hash table** proportional to the number of n-grams.

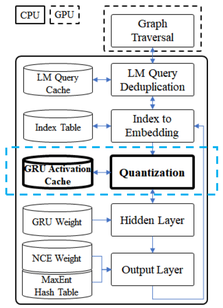
Baseline system – On-the-fly rescoring with cache



The LM queries with same history as well as following words are de-duplicated by applying a **cache strategy** at the start of the rescoring procedure. After the de-duplication, 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.

Proposed Two Methods

Quantization of history vectors



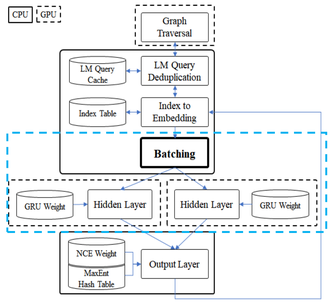
The current GRU hidden layer outputs computed based on the previous GRU hidden layer outputs (history vectors) which could be shared between **similar LM queries**. Therefore, in order to reuse the recomputed history vectors, we created **another cache for that vectors** just before computing RNNLM.

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.

CPU-GPU Hybrid Deployment of RNNLM Computation

▼ Only the middle layer of the RNNLM computations was deployed on the GPU side, the information needs to be shared across the two heterogeneous processor units **very frequently**. As the number of data exchanges increases, the decoding speed of the hybrid ASR system **inevitably decreases**.

▲ 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 embedding vectors 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.

Results

Redundancy rate according to quantization level

Precision	Count	Redundancy rate
(baseline)	103,904	0.00%
round-2	102,776	1.09%
round-1	102,776	1.09%
sign	**88,749**	**14.59%**

An extreme case of quantization where only signs of each element are stored, we were able to reduce **14.59%** of the computations.

Operation time for each RNNLM computation step(in secs)

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

The hidden layer takes **99%** of the overall computation. The computation is easily accelerated by GPUs. Thus we maximized the GPU utilization through a frame-wise batching strategy. (**10x** faster data transfer speed)

Performance on LibriSpeech's test sets

LM	Processor	Rescoring threads	Precision	dev-clean WER	dev-clean RTF	test-clean WER	test-clean RTF	dev-other WER	dev-other RTF	test-other WER	test-other 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

The proposed quantization strategy shows **1.23x faster** recognition speed compared to the baseline system. With three GPUs, we attained **real-time speech recognition** over all the test cases. The fastest average recognition speed was **0.72 RTF**.

SAMSUNG Research