Accurate and Compact Large Vocabulary Speech Recognition on Mobile Devices

Xin Lei[1] *Andrew Senior*[2] *Alexander Gruenstein*[1] *Jeffrey Sorensen*[2]

[1]Google Inc., Mountain View, CA USA
[2]Google Inc., New York, NY USA
{xinlei,andrewsenior,alexgru,sorenj}@google.com

Abstract

In this paper we describe the development of an accurate, small-footprint, large vocabulary speech recognizer for mobile devices. To achieve the best recognition accuracy, state-of-the-art deep neural networks (DNNs) are adopted as acoustic models. A variety of speedup techniques for DNN score computation are used to enable real-time operation on mobile devices. To reduce the memory and disk usage, on-the-fly language model (LM) rescoring is performed with a compressed n-gram LM. We were able to build an accurate and compact system that runs well below real-time on a Nexus 4 Android phone.

Index Terms: Deep neural networks, embedded speech recognition, SIMD, LM compression.

1. Introduction

Smartphones and tablets are rapidly overtaking desktop and laptop computers as people's primary computing device. They are heavily used to access the web, read and write messages, interact on social networks, etc. This popularity comes despite the fact that it is significantly more difficult to input text on these devices, predominantly by using an on-screen keyboard.

Automatic speech recognition (ASR) is a natural, and increasingly popular, alternative to typing on mobile sevices. Google offers the ability to search by voice [1] on Android, iOS, and Chrome; Apple's iOS devices come with Siri, a conversational assistant. On both Android and iOS devices, users can also speak to fill in any text field where they can type (see, *e.g.*, [2]), a capability heavily used to dictate SMS messages and e-mail.

A major limitation of these products is that speech recognition is performed on a server. Mobile network connections are often slow or intermittent, and sometimes non-existant. Therefore, in this study, we investigate techniques to build an accurate, small-footprint speech recognition system that can run in real-time on modern mobile devices.

Previously, speech recognition on handheld computers and smartphones has been studied in the DARPA sponsored Transtac Program, where speech-to-speech translation systems were developed on the phone [3, 4, 5]. In the Transtac systems, Gaussian mixture models (GMMs) were used to as acoustic models. While the task was a small domain, with limited training data, the memory usage in the resulting systems was moderately high.

In this paper, we focus on large vocabulary on-device dictation. We show that deep neural networks (DNNs) can provide large accuracy improvements over GMM acoustic models, with a significantly smaller footprint. We also demonstrate how memory usage can be significantly reduced by performing on-the-fly rescoring with a compressed language model during decoding.

The rest of this paper is organized as follows. In Section 2, the embedded GMM acoustic model is described. Section 3 presents the training of embedded DNNs, and the techniques we employed to speed up DNN inference at runtime. Section 4 describes the compressed language models for on-the-fly rescoring. Section 5 shows the experimental results of recognition accuracy and speed on Nexus 4 platform. Finally, Section 6 concludes the paper and discusses future work.

2. GMM Acoustic Model

Our embedded GMM acoustic model is trained on 4.2M utterances, or more than 3,000 hours of speech data containing randomly sampled anonymized voice search queries and other dictation requests on mobile devices. The acoustic features are 9 contiguous frames of 13-dimensional PLP features spliced and projected to 40 dimensions by linear discriminant analysis (LDA). Semi-tied covariances [6] are used to further diagonalize the LDA transformed features. Boosted-MMI [7] was used to train the model discriminatively.

The GMM acoustic model contains 1.3k clustered acoustic states, with a total of 100k Gaussians. To reduce model size and speed up computation on embedded platforms, the floating-point GMM model is converted to a fixed-point representation, similar to that described in [8]. Each dimension of the Gaussian mean vector is quantized into 8 bits, and 16-bit for precision vector. The resulting fixed-point GMM model size is about 1/3 of the floating-point model, and there is no loss of accuracy due to this conversion in our empirical testing.

3. DNNs for Embedded Recognition

We have previously described the use of deep neural networks for probability estimation in our cloud-based mobile voice recognition system [9]. We have adopted this system for developing DNN models for embedded recognition, and summarize it here.

The model is a standard feed-forward neural network with k hidden layers of n_h nodes, each computing a nonlinear function of the weighted sum of the outputs of the previous layer. The input layer is the concatenation of n_i consecutive frames of 40-dimensional log filterbank energies calculated on 25ms windows of speech every 10ms. The n_o softmax outputs estimate the posterior of each acoustic state. We have experimented with conventional logistic nonlinearities and rectified linear units that have recently shown superior performance in our large scale task [10], while also reducing computation.

While our server-based model has 50M parameters ($k = 4$, $n_h = 2560$, $n_i = 26$ and $n_o = 7969$), to reduce the memory and computation requirement for the embedded model, we experimented with a variety of sizes and chose $k = 6$, $n_h = 512$, $n_i = 16$ and $n_o = 2000$, or 2.7M parameters. The input window is asymmetric; each additional frame of future context adds 10ms of latency to the system so we limit ourselves to 5 future frames, and choose around 10 frames of past context, trading off accuracy and computation.

Our context dependency (CD) phone trees were initially constructed using a GMM training system that gave 14,247 states. By pruning this system using likelihood gain thresholds, we can choose an arbitrary number of CD states. We used an earlier large scale model with the full state inventory that achieved around 14% WER to align the training data, then map the 14k states to the desired smaller inventory. Thus we use a better model to label the training data to an accuracy that cannot be achieved with the embedded scale model.

3.1. Training

Training uses conventional backpropagation of gradients from a cross entropy error criterion. We use minibatches of 200 frames with an exponentially decaying learning rate and a momentum of 0.9. We train our neural networks on a dedicated GPU based system. With all of the data available locally on this system, the neural network trainer can choose minibatches and calculate the backpropagation updates.

3.2. Decoding speedup

Mobile CPUs are designed primarily for lower power usage and do not have as many or as powerful math units as CPUs used in server or desktop applications. This makes DNN inference, which is mathematically computationally expensive, a particular challenge. We exploit a number of techniques to speed up the DNN score computation on these platforms.

As described in [11], we use a fixed-point representation of DNNs. All activations and intermediate layer weights are quantized into 8-bit `signed char`, and biases are encoded as 32-bit `int`. The input layer remains floating-point, to better accommodate the larger dynamic ranges of input features. There is no measured accuracy loss resulting from this conversion to fixed-point format.

Single Instruction Multiple Data (SIMD) instructions are used to speed up the DNN computation. With our choice of smaller-sized fixed-point integer units, the SIMD acceleration is significantly more efficient, exploiting up to 8 way parallelism in each computation. We use a combination of inline assembly to speed up the most expensive matrix multiplication functions, and compiler intrinsics in the sigmoid and rectified linear calculations.

Batched lazy computation [11] is also performed. To exploit the multiple cores present on modern smartphones, we compute the activations up to the last layer in a dedicated thread. The output posteriors of the last layer are computed only when needed by the decoder in a separate thread. Each thread computes results for a batch of frames at a time. The choice of batch size is a tradeoff between computation efficiency and recognition latency.

Finally, frame skipping [12] is adopted to further reduce computation. Activations and posteriors are computed only every n_b frames and used for n_b consecutive frames. In experiments we find that for $n_b = 2$, the accuracy loss is negligible; however for $n_b \geq 3$, the accuracy degrades quickly.

4. Language Model Compression

We create n-gram language models appropriate for embedded recognition by first training a 1M word vocabulary and 18M n-gram Katz-smoothed 4-gram language model using Google's large-scale LM training infrastructure [13]. The language model is trained using a very large corpus (on the order of 20 billion words) from a variety of sources, including search queries, web documents and transcribed speech logs.

To reduce memory usage, we use two language models during decoding. First, a highly-pruned LM is used to build a small CLG transducer [14] that is traversed by the decoder. Second, we use a larger LM to perform on-the-fly lattice rescoring during search, similar to [15]. We have observed that a CLG transducer is generally two to three times larger than a standalone LM, so this rescoring technique significantly reduces the memory footprint.

Both language models used in decoding are obtained by shrinking the 1M vocabulary and 18M n-gram LM. We aggressively reduce the vocabulary to the 50K highest unigram terms. We then apply relative entropy pruning [16] as implemented in the OpenGrm toolkit [17]. The resulting finite state model for rescoring LM has 1.4M n-grams, with just 280K states and 1.7M arcs. The LM for first pass decoding contains only unigrams and about 200 bigrams.

We further reduce the memory footprint of the rescoring LM by storing it in an extremely memory-efficient manner, discussed below.

4.1. Succinct storage using LOUDS

If you consider a backoff language model's structure, the failure arcs from $(n + 1)$-gram contexts to n-gram contexts and, ultimately, to the unigram state form a tree. Trees can be stored using 2 bits per node using a level-order unary degree sequence (LOUDS), where we visit the nodes breadth-first writing 1s for the number of $(n+1)$-gram contexts and then terminating with a 0 bit. We build a bit sequence similarly for the degree of outbound non-ϕ arcs.

The LOUDS data structure provides *first-child*, *last-child*, and *parent* navigation, so we are able to store a language model without storing any next-state values. As a contiguous, index-free data object, the language model can be easily memory mapped.

The implementation of this model is part of the *OpenFst* library [18] and covered in detail in [19]. The precise storage requirements, measured in bits, are

$$4n_s + n_a + (W + L)(n_s + n_a) + Wn_f + c$$

where n_s is the number of states, n_f the number of final states, n_a is the number of arcs, L is the number of bits per word-id, and W is the number of bits per probability value. This is approximately one third the storage required by OpenFst's vector representation. For the models discussed here, we use 16 bits for both labels and weights.

During run time, to support fast navigation in the language model, we build additional indexes of the LOUDS bit sequences to support the operations $\mathrm{rank}_b(i)$ the number of b valued bits before index i, and its inverse $\mathrm{select}_b(r)$. We maintain a two level index that adds an additional $0.251(4n_s + n_a)$ bits. Here it is important to make use of fast assembly operations such as *find first set* during decoding, which we do through compiler intrinsics.

4.2. Symbol table compression

The word symbol table for an LM is used to map words to unique identifiers. Symbol tabels are another example of a data structure that can be represented as a tree. In this case we relied upon the implementation contained in the MARISA library [20].

This produces a symbol table that fits in just one third the space of the *concatenated strings* of the vocabulary, yet provides a bidirectional mapping between integers and vocabulary strings. We are able to store our vocabulary in about 126K bytes, less than 3 bytes per entry in a memory mappable image.

The MARISA library assigns the string to integer ids during compression, so we relabel all of the other components in our system to match this assignment.

5. Experimental Results

To evaluate accuracy performance, we use a test set of 20,000 anonymized transcribed utterances from users speaking in order to fill in text fields on mobile devices. This biases the test set towards dictation, as opposed to voice search queries, because dictation is more useful than search when no network connection is available.

To measure speed performance, we decode a subset of 100 utterances on an Android Nexus 4 (LG) phone. The Nexus 4 is equipped with a 1.5GHz quad-core Qualcomm Snapdragon S4 pro CPU, and 2GB of RAM. It runs the Android 4.2 operating system. To reduce start up loading time, all data files, including the acoustic model, the CLG transducer, the rescoring LM and the symbol tables are memory mapped on the device. We use a background thread to "prefetch" the memory mapped resources when decoding starts, which mitigates the slowdown in decoding for the first several utterances.

5.1. GMM acoustic model

The GMM configuration achieves a word error rate (WER) of 20.7% on this task, with an average real-time (RT) factor of 0.63. To achieve this speed, the system uses integer arithmetic for likelihood calculation and decoding. The Mahalanobis distance computation is accelerated using fixed-point SIMD instructions. Gaussian selection is used to reduce the burden of likelihood computation, and further efficiencies come from computing likelihoods for batches of frames.

5.2. Accuracy with DNNs

We compare the accuracy of DNNs with different configurations to the baseline GMM acoustic model in Table 1. A DNN with 1.48M parameters already outperforms the GMM in accuracy, with a disk size of only 17% of the GMM's. By increasing the number of hidden layers from 4 to 6 and number of outputs from 1000 to 2000, we obtain a large improvement of 27.5% relative in WER compared to the GMM baseline. The disk size of this DNN is 26% of the size of the GMMs.

For comparison, we also evaluate a server-sized DNN with an order of magnitude of more parameters, and it gives 12.3% WER. Note that all experiments in Table 1 use smaller LMs in decoding. In addition, with an un-pruned server LM, the server DNN achieves 9.9% WER while the server GMM achieves 13.5%. Therefore, compared to a full-size DNN server system, there is a 2.4% absolute loss due to smaller LMs, and 2.8% due to smaller DNN. Compared to the full-size GMM server system, the embedded DNN system is about 10% relatively worse in WER.

The impact of frame skipping is evaluated with the DNN_6×512 model. As shown in Table 2, the accuracy performance quickly degrades when n_b is larger than 2.

Table 2: *Accuracy results with frame skipping in a DNN system.*

n_b	1	2	3	4	5
WER (%)	15.1	15.2	15.6	16.0	16.7

5.3. Speed benchmark

For speed benchmark, we measure average RT factor as well as 90-percentile RT factor. As shown in Table 3, the baseline GMM system with SIMD optimization gives an average RT factor of 0.63. The fixed-point DNN gives 1.32×RT without SIMD optimization, and 0.75×RT with SIMD. Batched lazy computation improves average RT by 0.06 but degrades the 90-percentile RT performance, probably due to less efficient on-demand computation for difficult utterances. After frame skipping with $n_b = 2$, the speed of DNN system is further improved slightly to 0.66×RT. Finally, the overhead of the compact LOUDS based LM is about 0.13×RT on average.

Table 3: *Averge real-time (RT) and 90-percentile RT factors of different system settings.*

	Average RT	RT(90)
GMM	0.63	0.90
DNN (fixed-point)	1.32	1.43
+ SIMD	0.75	0.87
+ lazy batch	0.69	1.01
+ frame skipping	0.66	0.97
+ LOUDS	0.79	1.24

5.4. System Footprint

Compared to the baseline GMM system, the new system with LM compression and DNN acoustic model achieves a much smaller footprint. The data files sizes are listed in Table 4. Note that conversion of the 34MB floating-point GMM model to a 14MB fixed-point GMM model itself provides a large reduction in size.

The use of DNN reduces the size by 10MB, and the LM compression contributed to another 18MB reduction. Our final embedded DNN system size is reduced from 46MB to 17MB, while achieving a big WER reduction from 20.7% to 15.2%.

6. Conclusions

In this paper, we have described a fast, accurate and small-footprint speech recognition system for large vocabulary dictation on the device. DNNs are used as acoustic model, which provides a 27.5% relative WER improvement over the baseline GMM models. The use of DNNs also significantly reduces the memory usage. Various techniques are adopted to speed up the DNN inference at decoding time. In addition, a LOUDS based language model compression reduces the rescoring LM size by more than 60% relative. Overall, the size of the data files of the system is reduced from 46MB to 17MB.

Table 1: *Comparison of GMM and DNNs with different sizes. The input layer is denoted by number of filterbank energies × the context window size (left + current + right). The hidden layers are denoted by number of hidden layers × number of nodes per layer. The number of outputs is the number of HMM states in the model.*

Model	WER (%)	Input Layer	Hidden Layers	# Outputs	# Parameters	Size
GMM	20.7	-	-	1314	8.08M	14MB
DNN_4×400	22.6	40×(8+1+4)	4×400	512	0.9M	1.5MB
DNN_4×480	20.3	40×(10+1+5)	4×480	1000	1.5M	2.4MB
DNN_6×512	15.1	40×(10+1+5)	6×512	2000	2.7M	3.7MB
Server DNN	12.3	40×(20+1+5)	4×2560	7969	49.3M	50.8MB

Table 4: *Comparison of data file sizes (in MB) in baseline GMM system and DNN system with and without LOUDS LM compression. AM denotes acoustic model, CLG is the transducer for decoding, LM denotes the rescoring LM, and symbols denote the word symbol table.*

System	AM	CLG	LM	Symbols	Total
GMM	14	2.7	29	0.55	46
+ LOUDS	14	2.7	10.7	0.13	27
DNN	3.7	2.8	29	0.55	36
+ LOUDS	3.7	2.8	10.7	0.13	17

Future work includes speeding up rescoring using the LOUDS LM as well as further compression techniques. We also continue to investigate the accuracy performance with different sizes of LM for CLG and rescoring.

7. Acknowledgements

The authors would like to thank our former colleague Patrick Nguyen for implementing the portable neural network runtime engine used in this study. Thanks also to Vincent Vanhoucke and Johan Schalkwyk for helpful discussions and support during this work.

8. References

[1] J. Schalkwyk, D. Beeferman, F. Beaufays, B. Byrne, C. Chelba, M. Cohen, M. Kamvar, and B. Strope, "Google search by voice: A case study," in *Advances in Speech Recognition: Mobile Environments, Call Centers and Clinics*, pp. 61–90. Springer, 2010.

[2] B. Ballinger, C. Allauzen, A. Gruenstein, and J. Schalkwyk, "On-demand language model interpolation for mobile speech input," in *Proc. Interspeech*, 2010.

[3] J. Zheng et al., "Implementing SRI's Pashto speech-to-speech translation system on a smart phone," in *SLT*, 2010.

[4] J. Xue, X. Cui, G. Daggett, E. Marcheret, and B. Zhou, "Towards high performance LVCSR in speech-to-speech translation system on smart phones," in *Proc. Interspeech*, 2012.

[5] R. Prasad et al., "BBN Transtalk: Robust multilingual two-way speech-to-speech translation for mobile platforms," *Computer Speech and Language*, vol. 27, pp. 475–491, February 2013.

[6] M. J. F. Gales, "Semi-tied covariance matrices for hidden Markov models," *IEEE Trans. Speech and Audio Processing*, vol. 7, pp. 272–281, 1999.

[7] D. Povey, D. Kanevsky, B. Kingsbury, B. Ramabhadran, G. Saon, and K. Visweswariah, "Boosted MMI for model and feature-space discriminative training," in *Proc. ICASSP*, 2008.

[8] E. Bocchieri, "Fixed-point arithmetic," *Automatic Speech Recognition on Mobile Devices and over Communication Networks*, pp. 255–275, 2008.

[9] N. Jaitly, P. Nguyen, A. W. Senior, and V. Vanhoucke, "Application of pretrained deep neural networks to large vocabulary speech recognition," in *Proc. Interspeech*, 2012.

[10] M. D. Zeiler et al., "On rectified linear units for speech processing," in *Proc. ICASSP*, 2013.

[11] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, 2011.

[12] V. Vanhoucke, M. Devin, and G. Heigold, "Multiframe deep neural networks for acoustic modeling," in *Proc. ICASSP*, 2013.

[13] T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean, "Large language models in machine translation," in *EMNLP*, 2007, pp. 858–867.

[14] M. Mohri, F. Pereira, and M. Riley, "Speech recognition with weighted finite-state transducers," *Handbook of Speech Processing*, pp. 559–582, 2008.

[15] T. Hori and A. Nakamura, "Generalized fast on-the-fly composition algorithm for WFST-based speech recognition," in *Proc. Interspeech*, 2005.

[16] A. Stolcke, "Entropy-based pruning of backoff language models," in *DARPA Broadcast News Transcription and Understanding Workshop*, 1998, pp. 8–11.

[17] B. Roark, R. Sproat, C. Allauzen, M. Riley, J. Sorensen, and T. Tai, "The OpenGrm open-source finite-state grammar software libraries," in *Proceedings of the ACL 2012 System Demonstrations*. 2012, ACL '12, pp. 61–66, Association for Computational Linguistics.

[18] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFst: A general and efficient weighted finite-state transducer library," in *Proceedings of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007)*. 2007, vol. 4783 of *Lecture Notes in Computer Science*, pp. 11–23, Springer, http://www.openfst.org.

[19] J. Sorensen and C. Allauzen, "Unary data structures for language models," in *Proc. Interspeech*, 2011.

[20] S. Yata, "Prefix/Patricia trie dictionary compression by nesting prefix/Patricia tries (japanese)," in *Proceedings of 17th Annual Meeting of the Association for Natural Language*, Toyohashi, Japan, 2011, NLP2011, https://code.google.com/p/marisa-trie/.