PocketSUMMIT: Small-Footprint Continuous Speech Recognition

I. Lee Hetherington

MIT Computer Science and Artificial Intelligence Laboratory
Cambridge, Massachusetts, USA

ilh@csail.mit.edu

Abstract

We present PocketSUMMIT, a small-footprint version of our SUMMIT continuous speech recognition system. With portable devices becoming smaller and more powerful, speech is increasingly becoming an important input modality on these devices. PocketSUMMIT is implemented as a variable-rate continuous density hidden Markov model with diphone context-dependent models. We explore various Gaussian parameter quantization schemes and find 8:1 compression or more is achievable with little reduction in accuracy. We also show how the quantized parameters can be used for rapid table lookup. We explore first-pass language model pruning in a finite-state transducer (FST) framework, as well as FST and n-gram weight quantization and bit packing, to further reduce memory usage. PocketSUMMIT is currently able to run a moderate vocabulary conversational speech recognition system in real time in a few MB on current PDAs and smart phones.

Index Terms: speech recognition, small footprint, parameter quantization, finite-state transducer

1. Introduction

As technology improves, portable devices such as PDAs and mobile phones are becoming smaller and more computationally powerful. We wish to do more with these devices, yet the smaller form factor generally makes them more difficult to use. Speech interfaces may be able to harness the computational power and provide a more intuitive interface despite minimal screen and keyboard sizes.

Historically, our SUMMIT automatic speech recognition system was developed to support research by providing a flexible framework. As long as it could support real-time recognition within a conversational system running on a Unix workstation, memory use and additional speed was not a priority. When confronted with portable devices with modest memory and processing resources, it was clear we needed to simplify and rewrite most of SUMMIT to create PocketSUMMIT.

We initially targeted PocketPC and smart phone devices running Windows CE/Mobile, which typically use a 400–600MHz ARM processor, no floating-point hardware, and 32–64MB RAM. The lack of floating-point hardware meant that all significant computation needed to use fixed-point (integer) operations. With limited memory and typically slow memory access speed, we strove to push the footprint of PocketSUMMIT down to a few MB. Aggressive scalar quantization was used on acoustic model parameters, finite-state transducer (FST) weights, and n-gram parameters to reduce memory footprint without hurting accuracy. Even though FST and rescoring n-gram data structures were aggressively bit packed down to minimal size and accessed frequently in the main decoder search, we were pleasantly surprised to find that not only did this bit packing not hurt speed, it actually improved it.

This research is sponsored by the T-Party Project, a joint research program between MIT and Quanta Computer Inc., Taiwan.

Porting existing automatic speech recognition engines to mobile devices is becoming increasingly popular. Examples include PocketSphinx [1] and systems from AT&T [2] and IBM [3]. Our use of Gaussian parameter quantization was inspired by work at Nokia [4] for use on mobile phones.

2. PocketSUMMIT

In this section we describe PocketSUMMIT, and in particular how it differs from our baseline SUMMIT recognition system. Throughout this section, we used a 2000-word weather information domain (Jupiter) [5] to benchmark progress in terms of memory use and recognition accuracy, using a test set of 1711 utterances collected over the telephone from real users.

2.1. Variable-rate HMM

The first simplification to our baseline system was to use only landmark features, rather than the combination of more complex segmental and landmark features [6]. 14 Mel-frequency cepstral coefficients (MFCCs) are computed every 5 ms, and a landmark detector hypothesizes likely phonetic boundaries, averaging about 30 ms per landmark. A diphone context-dependent hidden Markov model (HMM) is used to model acoustic features centered at each landmark. The landmark rate represents about a three-fold reduction compared to that typically used in other HMM systems.

Figure 1 shows the two-state HMM topology used for each diphone. The transition from phone a to phone b is modelled by $t(a|b)$, and internal landmarks are modeled by $i(b)$. The internal landmarks are those that have been hypothesized between phone boundaries. In general, the internal model can be context-dependent as well, but we used context-independent internal models in this work. The landmark feature vector consists of averages over eight regions around a landmark, the furthest out to ± 75 ms. The resulting 112 dimensions are passed through principal component analysis to reduce dimensionality to 50, reduce statistical dependence, and to normalize variance.

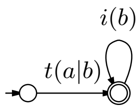

Figure 1: Diphone HMM topology for diphone $a|b$.

Later, in Section 2.2.1 we describe how to take advantage of these properties by using the same scalar quantizer for all dimensions and a single Gaussian table.

2.2. Acoustic modeling

The distributions of models $t(a|b)$ and $i(b)$ are modeled using Gaussian mixtures with diagonal covariance matrices, with a maximum of 50 and average of 22 mixtures per model.

2.2.1. Parameter quantization

The primary motivation for quantizing Gaussian parameters was to reduce the memory footprint of the acoustic models as done in [4]. As an example, our baseline SUMMIT recognizer for the weather domain utilizes 1,365 diphone acoustic models and a total of nearly 30,000 diagonal Gaussians of dimensionality 50. Storing all mean μ and variance σ^2 parameters as 32-bit floating-point values requires about 12 MB. By quantizing each of these parameters, we can greatly reduce the memory footprint. We quantized the parameters from trained floating-point models and evaluated the effect on recognition accuracy.

We used scalar quantization, where each dimension of a mean vector $\vec{\mu}$, variance vector $\vec{\sigma}^2$, and feature vector $\vec{x}$, is quantized independently. We compared both linear and non-linear quantization, the latter trained using the iterative Lloyd-Max algorithm [7]. As we will see in Section 2.2.2, there are computational reasons why we may prefer linear quantization of the mean and feature values.

We trained our quantizers on the μ and σ^2 parameters contained within our models. We found that, due to the principal component analysis and variance normalization performed on our feature vectors, there was no disadvantage to using the same quantizer across different dimensions. We achieve slightly better quantization results training the variance quantizer on $1/\sigma$ as opposed to σ, as that caused less distortion for more sensitive low-variance Gaussians.

Table 1 summarizes our best non-linear mean μ and variance σ^2 quantization results for various compression ratios, with the same quantizers applied across all feature dimensions. For each level of compression, we show the best-performing breakdown in bits for μ and σ^2. We were surprised to see that, even at 8:1 compression (5 bits for μ and 3 bits for σ^2), recognition accuracy was not significantly degraded with respect to the baseline floating-point parameters. The 8:1 compression ratio is particularly interesting because it means we can fit a single dimension's (μ, σ^2) pair within one byte, which has computational advantages, as presented in Section 2.2.2.

We also examined linear quantization of the mean μ parameters. There are computational advantages to such quantization with respect to Gaussian table lookup. A 5-bit linear quantizer with its minimum and maximum values set to the 0.5% and 99.5% quantiles performed the best. Combining the linear 5-bit mean quantizer and the 3-bit non-linear variance quantizer, we were able to achieve a 9.4% word error rate.

We did not consider vector or sub-vector quantization of the feature space or Gaussian parameters at this time. Such quantization requires vector operations (e.g., dot product), and for large codebooks, memory and computation may exceed that of scalar quantization. We may explore such techniques in the future, although due to our "whitened" feature space, there may not be much covariance structure to exploit.

bits		compression	WER
μ	σ		
32	32	1:1	9.5%
8	8	4:1	9.6%
6	4	6:1	9.3%
5	3	8:1	9.6%
4	2	10:1	10.1%
2	2	16:1	13.7%

Table 1: Gaussian parameter scalar quantization.

2.2.2. Computation

We now examine how quantized Gaussian parameters can be used to evaluate multivariate Gaussians using table lookup. The well-known density for a mixture of diagonal Gaussians of dimensionality D is

$$P(x|c) = \sum_{m \in M_c} w_m \prod_{d=1}^{D} \frac{\exp\left(-(x_d - \mu_{m,d})^2/\sigma_{m,d}^2\right)}{\sigma_{m,d}\sqrt{2\pi}} \tag{1}$$

where M_c is the set of mixtures for class c, and w_m is the mixture weight for mixture component m. Switching to log probabilities, this becomes

$$\log P(x|c) = \log \sum_{m \in M_c} \exp\left[\log w_m + G_m(x)\right] \tag{2}$$

where $G_m(x)$ is the log density of mixture component m, which can be computed as

$$G_m(x) = \sum_{d=1}^{D} \left[\frac{-(x_d - \mu_{m,d})^2}{\sigma_{m,d}^2} - \log\left(\sigma_{m,d}\sqrt{2\pi}\right) \right] \tag{3}$$

By quantizing x to $\tilde{x}$, μ to $\tilde{\mu}$, and σ to $\tilde{\sigma}$, we can approximate $G_m(x)$ by a sum of table lookups

$$G_m(x) \approx \sum_{d=1}^{D} T\left[\tilde{x}_d, \tilde{\mu}_{m,d}, \tilde{\sigma}_{m,d}\right] \tag{4}$$

The total number of table entries is $Q_x Q_\mu Q_\sigma$, where Q_i is the number of discrete quantized values for parameter i.

However, considering the form of (3), if we use the same linear quantizer for both x and μ we can further simplify this to

$$G_m(x) \approx \sum_{d=1}^{D} T\left[|\tilde{x}_d - \tilde{\mu}_{m,d}|, \tilde{\sigma}_{m,d}\right] \tag{5}$$

The total number of table entries in T is now only $Q_\mu Q_\sigma$. If μ and σ are quantized so they together fit in a single byte (e.g., 5 bits for $\tilde{\mu}$ and 3 bits for $\tilde{\sigma}$), we need only 256 total table entries for all possible Gaussian density computations.

Another advantage to $\tilde{\mu}$ and $\tilde{\sigma}$ fitting in a single byte is that we can use 32-bit operations to load $\tilde{x}$ and $(\tilde{\mu}, \tilde{\sigma})$, compute $|\tilde{x} - \tilde{\mu}|$, and compute indices into table T four dimensions at a time. Even without using specialized MMX-style operations, which may or may not be available on a given mobile processor, we found we could evaluate acoustic model densities nearly twice as fast by operating on four dimensions at a time vs. only one at a time.

Although the 8:1 compression of Gaussian parameters and table lookup implementation was targeted at mobile devices with limited memory and no floating-point hardware, we were pleasantly surprised to find these techniques can improve speed on more powerful desktop hardware. Our baseline system utilizes hand-crafted SSE code for x86 hardware to operate on four floating-point dimensions at a time, but the new fixed-point implementation yielded a significant 35–70% speed gain.

Note that, although our baseline system can make use of mixture tying and Gaussian selection to speed model evaluation, we have not yet implemented these within PocketSUMMIT. Tying is very straightforward, but Gaussian selection can involve a significant memory cost to store overlapping lists of Gaussians to evaluate for each feature space region. In the future, we plan to investigate techniques similar to [3] to address this issue.

2.3. Finite-state transducer (FST) formulation

We have long utilized finite-state transducers (FSTs) within our SUMMIT system for combining and optimizing various linguistic constraints [5, 8]. We typically make use of two FSTs in different recognition passes. The first-pass FST is

$$R = M \circ C \circ P \circ L \circ G_1 \tag{6}$$

where G_1 is the first-pass language model, L is the lexicon modeling words as sequences of phonemes, P applies a phonological model from phoneme sequences to phone sequences, C applies context-dependent phone model labels, and M represents the individual model topology for each context-dependent model (e.g., Figure 1). These individual FSTs are typically composed together and then optimized using ϵ-removal, determinization, and minimization. The result is a single, flat FST that can be rapidly traversed during the initial recognition pass.

The second FST G_2 applies the final language model G to the output of the first pass. The idea is that a much smaller language model G_1 can be used in the first stage of decoding and the much larger full language model G can be used in a later stage. If we define the incremental n-gram $G_2 = G_1^{-1} \circ G$, then G_2 can be composed with the output of the first pass to rescore with the language model G. Typically, G_2 can be formed by subtracting G_1's log probabilities from those in G.

2.3.1. FST size reduction by pruning

Typically, we use a pruned bigram for G_1 and a full trigram for G. By varying the amount of pruning applied to G_1, we can vary the size of the recognition FST R. As we reduce the size of G_1, we are using a weaker language model in the first pass, and thus may introduce search errors if correct hypotheses fail to survive beam pruning. In [9], we also found that FST determinization of R performed a kind of tree-based language model lookahead based on G_1, which can be helpful with aggressive beam pruning.

Table 2 shows how recognition accuracy degrades as we increasingly prune G_1, along with resulting bigram and FST sizes. Here we prune the bigram G_1 by applying a threshold to bigram counts. An alternative technique that yields slightly better results is Stolcke pruning [10]. With a threshold of 4, we can reduce the FST size by more than a factor of two with only a small degradation in accuracy. Note that the full language model G is applied in all cases during N-best rescoring.

2.3.2. FST weight quantization

In PocketSUMMIT, to further reduce the size of the recognition FST R and incremental n-gram G_2, we applied separate non-linear quantization to the weights of each. Table 3 summarizes the effects on accuracy of various quantization on R and G_2. It is interesting to note that we can more aggressively quantize the weights in G_2 compared with those in R. We hypothesize that, because G_2's weights represent n-gram log probability differences spanning a smaller range, they do not require as many quantization levels. We settled on 5-bit quantization for FST R weights and 3-bit quantization for n-gram G_2.

threshold	bigrams	R transitions	WER
1	23411	259164	9.5%
2	12964	171822	9.6%
4	7748	124998	9.9%
8	4794	98670	10.2%
16	2997	22404	10.7%

Table 2: Recognition accuracy vs. FST R size.

bits	WER
32	9.5%
7	9.8%
6	9.6%
5	9.8%
4	9.8%
3	10.2%

(a) FST

bits	WER
32	9.5%
6	9.9%
5	9.8%
4	9.8%
3	9.7%
2	10.1%

(b) n-gram

bits		WER
FST	n-gram	
32	32	9.5%
6	3	9.7%
5	3	9.6%
4	3	9.8%
3	3	10.3%

(c) FST and n-gram

Table 3: FST R and incremental n-gram G_2 weight quantization.

2.3.3. FST bit packing

In the baseline SUMMIT system, the FST R is stored directly with a transition consisting of five 32-bit values (previous state index, next state index, input label index, output label index, and weight) for a total of 160 bits per transition. In PocketSUMMIT, we chose to bit pack transitions utilizing a variable number of bits in order to exploit FST structure. For example, self loops will only contain $i(b)$ model input labels, not have word output labels or weights, and the next state does not need to be represented explicitly. The result is that transitions can be encoded with 12–44 bits, with an average of about 28 bits/transition. With 125,000 transitions, this resulted in a size reduction from 5.9 MB to only 544 kB. We apply similar bit-packing to the n-gram G_2, reducing its size for 96,000 trigrams from 1.0 MB to 352 kB.

We were a bit concerned that using a bit-packed representation of FST R, in which the transitions are variable-sized, would adversely affect recognition speed when comparing against fixed-size data structures using properly aligned 32-bit values. After all, this FST is heavily accessed during the first pass to look up partial path extensions. However, it has been our experience that even though bit-level operations are needed to unpack the variable-sized FST structure, overall speed is approximately 5–10% *faster* with the smaller, bit-packed and weight-quantized representation. Presumably, the smaller FST representation fits better within processor caches, more than making up for the operations needed to unpack it on demand.

2.4. Decoder search

In our baseline SUMMIT system, recognition is spread over several passes. Two passes are used to generate a state/word graph using FST R, and two more passes are used to rescore it with G_2 and compute the final N-best list (or word graph) result.

For PocketSUMMIT, we simplified the decoder to produce a state/word graph in the first pass. The beam pruning parameters directly control the size of this graph. When the pool of pre-allocated graph states or transitions is exhausted, the graph is incrementally pruned using a backwards A* search to keep the most likely transitions. At the end of the utterance, a deep N-best list is computed with an A* search, and finally this N-best list is rescored with the incremental n-gram G_2. The new decoder search uses considerably less memory and runs much faster, with typical latency under 10 ms on desktop hardware.

2.5. Putting it all together

We already had a fixed-point front-end to compute MFCCs from earlier distributed speech recognition work [11]. All other components of PocketSUMMIT were rewritten using only fixed-point arithmetic. The external models (Gaussian mixture models, recognition FST R, and rescoring n-gram G_2) are all implemented using memory-mapped files so they can be loaded on demand by the operating system. The PocketSUMMIT continuous speech recognition engine compiles to a small 130 kB Windows DLL or Linux shared library.

For the Jupiter weather information domain, the memory-mapped model components total 2.5 MB. Using more recent acoustic models trained using the minimum classification error (MCE) criterion [12], we have since halved the number of Gaussian mixtures, reducing this total to 1.7 MB with no degradation in accuracy. The total run-time memory footprint, including memory-mapped components, DLL, and dynamic memory is 3.2 MB in Windows CE/Mobile. The recognizer currently runs on a 408 MHz ARM-based Windows Mobile smart phone at approximately real time speed.[1] The word error rate is only slightly degraded with resepect to the baseline system at 9.9%.

PocketSUMMIT can also run from a context-free or finite-state grammar. We have put together a system one might use to access music on a personal music device. Loading the music content from such a device containing 2600 song titles, 350 albums, 330 artists, and a total vocabulary of about 3000 words, PocketSUMMIT runs in under 2.5 MB and nearly real-time speed on a 204 MHz ARM. In this case, PocketSUMMIT's footprint is smaller than a typical song's MP3 file.

3. Conclusion and future work

We have completed an initial implementation of PocketSUMMIT by simplifying our baseline SUMMIT system, aggressively quantizing Gaussian, FST, and n-gram parameters, and generally rewriting the complete decoder for fixed-point operations with an eye toward minimizing memory footprint at every opportunity. Particular care was given to efficient computation of the Gaussian mixture models. Initial indications are that PocketSUMMIT can easily run moderate-vocabulary conversational speech recognition at real-time speed on typical PDA and smart phone processors with a memory footprint of a few MB. Properties of PocketSUMMIT that we believe are particularly helpful in reducing computation include the variable-rate landmark detection, simple diphone HMM topology, efficient acoustic model compression and evaluation, and the use of FSTs to flexibly optimize constraints used in the first pass.

Future work will include implementing some of the capabilities of SUMMIT left out of the initial implementation of PocketSUMMIT including dialogue state-dependent grammar/vocabulary and multi-pass dynamic vocabulary recognition [13], mixture tying, and Gaussian selection. It appears that our MCE-trained Gaussian mixture models are more sensitive to parameter quantization when compared to ML-trained models, and we intend to examine and address the interaction of MCE training with parameter quantization.

[1]PocketSUMMIT runs Jupiter at approximately 20 times faster than real time on a 3.2 GHz Pentium 4 Linux system, in the same 3.2 MB. This speed is about 5 times faster than the baseline SUMMIT.

4. Acknowledgements

Many thanks to Bo-June (Paul) Hsu for his implementation of a GUI and general help with Windows CE/Mobile.

5. References

[1] D. Huggins-Daines *et al.*, "PocketSphinx: a free, real-time continuous speech recognition system for handheld devices," *Proc. ICASSP*, pp. 185–188, Toulouse, May 2006.

[2] E. Bocchieri and D. Blewett, "A decoder for LVCSR based on fixed-point arithmetic," *Proc. ICASSP*, pp. 1113–1116, Toulouse, May 2006.

[3] M. Novak, "Towards large vocabulary ASR on embedded platforms," *Proc. Interspeech*, Oct. 2004.

[4] M. Vasilache, "Speech recognition using HMMs with quantized parameters," *Proc. ICSLP*, pp. 441–444, Beijing, Oct. 2000.

[5] V. Zue *et al.*, "Jupiter: a telephone-based conversational interface for weather information," *IEEE Trans. Speech and Audio Proc.*, 8(1), Jan. 2000.

[6] J. Glass, "A probabilistic framework for segment-based speech recognition," *Computer Speech and Language*, 17:137–152, 2003.

[7] J. Max, "Quantizing for minimal distortion," *Trans. on Info. Theory*, 6(1):7–12, Mar. 1960.

[8] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, 16(1):69–88, Jan. 2002.

[9] H. Dolfing and I. Hetherington, "Incremental language models for speech recognition using finite-state transducers," *Proc. IEEE ASRU*, Madonna di Campiglio, Dec. 2001.

[10] A. Stolcke, "Entropy-based pruning of backoff language models," *Proc. DARPA Broadcast News Transcription and Understanding Workshop*, 1998.

[11] L. Miyakawa, *Distributed Speech Recognition within a Segment-Based Framework*, S.M. Thesis, MIT Dept. of EECS, Jun. 2003.

[12] E. McDermott and T. Hazen, "Minimum classification error training of landmark models for real-time continuous speech recognition," *Proc. ICASSP*, pp. 937–940, Montreal, May 2004.

[13] L. Hetherington, "A multi-pass, dynamic-vocabulary approach to real-time, large-vocabulary speech recognition," *Proc. Interspeech*, pp. 545–548, Lisbon, Sep. 2005.