Carnegie Mellon University
Research Showcase @ CMU

Computer Science Department | School of Computer Science

2008

Mixture Pruning and Roughening for Scalable Acoustic Models

Alexander I. Rudnicky
Carnegie Mellon University, ar28@andrew.cmu.edu

David Huggins-Daines
Carnegie Mellon University

Follow this and additional works at: http://repository.cmu.edu/compsci

Published In
.

This Conference Proceeding is brought to you for free and open access by the School of Computer Science at Research Showcase @ CMU. It has been accepted for inclusion in Computer Science Department by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

Mixture Pruning and Roughening for Scalable Acoustic Models

David Huggins-Daines
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
dhuggins@cs.cmu.edu

Alexander I. Rudnicky
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
air@cs.cmu.edu

Abstract

In an automatic speech recognition system using a tied-mixture acoustic model, the main cost in CPU time and memory lies not in the evaluation and storage of Gaussians themselves but rather in evaluating the mixture likelihoods for each state output distribution. Using a simple entropy-based technique for pruning the mixture weight distributions, we can achieve a significant speedup in recognition for a 5000-word vocabulary with a negligible increase in word error rate. This allows us to achieve real-time connected-word dictation on an ARM-based mobile device.

1 Introduction

As transistors shrink and CPUs become faster and more power-efficient, we find ourselves entering a new age of intelligent mobile devices. We believe that not only will these devices provide access to rich sources of on-line information and entertainment, but they themselves will find new applications as personal knowledge management agents. Given the constraints of the mobile form factor, natural speech input is crucial to these applications. However, despite the advances in processor technology, mobile devices are still highly constrained by their memory and storage subsystems.

2 Semi-Continuous Acoustic Models

Recent research into acoustic model compression and optimization of acoustic scoring has focused on "Fully Continuous" acoustic models, where each Hidden Markov Model state's output probability distribution is modeled by a mixture of multivariate Gaussian densities. This type of model allows large amounts of training data to be efficiently exploited to produce detailed models. However, due to the large number of parameters in these models, approximate computation techniques (Woszczyna, 1998) are required in order to achieve real-time recognition even on workstation-class hardware.

Another historically popular type of acoustic model is the so-called "Semi-Continuous" or tied-mixture model, where a single codebook of Gaussians is shared by all HMM states (Huang, 1989). The parameters of this codebook are updated using the usual Baum-Welch equations during training, using sufficient statistics from all states. The mixture weight distributions therefore become the main source of information used to distinguish between different speech sounds.

There are several benefits to semi-continuous models for efficient speech recognition. The most obvious is the greatly reduced number of Gaussian densities which need to be computed. With fully continuous models, we must compute one codebook of 16 or more Gaussians for each HMM state, of which there may be several thousand active for any given frame of speech input. For semi-continuous models, there is a single codebook with a small number of Gaussians, typically 256. In addition, since only a few Gaussians will have non-negligible densities for each frame of speech, and this set of Gaussians tends to change slowly, partial computation of each density is possible.

Another useful property of semi-continuous models is that the mixture weights for each state have the form of a multinomial distribution, and are thus amenable to various smoothing and adaptation techniques. In particular, Bayesian and quasi-Bayes

adaptation (Huo and Chan, 1995) are effective and computationally efficient.

3 Experimental Setup

All experiments in this paper were performed using PocketSphinx (Huggins-Daines et al., 2006). The baseline acoustic model was trained from the combined WSJ0 and WSJ1 "long" training sets (Paul and Baker, 1992), for a total of 192 hours of speech. This speech was converted to MFCC features using a bank of 20 mel-scale filters spaced from 0 to 4000Hz, allowing the model to work with audio sampled at 8kHz, as is typical on mobile devices. We used 5-state Hidden Markov Models for all phones. Output distributions were modeled by a codebook of 256 Gaussians, shared between 5000 tied states and 220 context-independent states. Only the first pass of recognition (static lexicon tree search) was performed.

Our test platform is the Nokia N800, a handheld Internet Tablet. It uses a Texas Instruments OMAP^{TM} 2420 processor, which combines an ARM11 RISC core and a C55x DSP core on a single chip. The RISC core is clocked at 400MHz while the DSP is clocked at 220MHz. In these experiments, we used the ARM core for all processing, although we have also ported the MFCC extraction code to the DSP. The decoder binaries, models and audio files were stored on a high-speed SD flash card formatted with the `ext3` journaling filesystem. Using the standard `bcb05cnp` bigram language model, we obtained a baseline word error rate of 9.46% on the `si_et_05` test set. The baseline performance of this platform on the test set is 1.40 times real-time, that is, for every second of speech, 1.40 seconds of CPU time are required for recognition.

We used the `oprofile` utility[1] on the Nokia N800 to collect statistical profiling information for a subset of the test corpus. The results are shown in Table 1. We can see that three operations occupy the vast majority of CPU time used in decoding: managing the list of active HMM states, computing the codebook of Gaussians, and computing mixture densities.

The size of the files in the acoustic model is shown in Table 2. The amount of CPU time required to calculate densities is related to the size of the mixture weight distribution by the fact that the N800 has a single-level 32Kb data cache, while a typical desktop processor has two levels of cache totalling at least 1Mb. We used `cachegrind`[2] to simulate the memory hierarchy of an OMAP versus an AMD K8 desktop processor with 64Kb of L1 cache and 512Kb of L2 cache, with results shown in Table 3.

While other work on efficient recognition has focused on quantization of the Gaussian parameters (Leppänen and Kiss, 2005), in a semi-continuous model, the number of these parameters is small

Function	%CPU
HMM evaluation	**22.41**
hmm_vit_eval_5st_lr	13.36
hmm_vit_eval_5st_lr_mpx	3.71
Mixture Evaluation	**21.66**
get_scores4_8b	14.94
fast_logmath_add	6.72
Lexicon Tree Search	**19.89**
last_phone_transition	5.25
prune_nonroot_chan	4.15
Active List Management	**15.57**
hmm_sen_active	13.75
compute_sen_active	1.19
Language Model Evaluation	**7.80**
find_bg	2.55
ngram_ng_score	2.13
Gaussian Evaluation	**5.87**
eval_cb	5.59
eval_topn	0.28
Acoustic Feature Extraction	**3.60**
fe_fft_real	1.59
fixlog2	0.77

Table 1: CPU profiling, OMAP platform

File	Size (bytes)
`sendump` (mixture weights)	5345920
`mdef` (triphone mappings)	1693280
`means` (Gaussians)	52304
`variances` (Gaussians)	52304
`transition_matrices`	5344

Table 2: File sizes, WSJ1 acoustic model

[1] `http://oprofile.sourceforge.net/`

[2] `http://valgrind.org/`

Function	ARM	K8
hmm_vit_eval_5st_lr	4.71	3.95
hmm_sen_active	3.55	3.76
get_scores4_8b	2.87	1.92
prune_root_chan	2.07	2.29
prune_nonroot_chan	1.99	1.73
eval_cb	1.73	1.77
hmm_vit_eval_5st_lr_mpx	1.30	0.80

Table 3: Data cache misses (units of 10^7)

enough that little cost is incurred by storing and calculating them as 32-bit fixed-point numbers. Therefore, we focus here on ways to reduce the amount of storage and computation used by the mixture weight distributions.

4 Mixture Roughening

Our method for speeding up mixture computation is based on the observation that mixture weight distributions are typically fairly "spiky", with most of the probability mass concentrated in a small number of mixture weights. One can quantify this by calculating the perplexity of the mixture distributions:

$$pplx(w_i) = \exp \sum_{k=0}^{N} w_{ik} \log \frac{1}{w_{ik}}$$

A histogram of perplexities is shown in Figure 1. The perplexity can be interpreted as the average number of Gaussians which were used to generate an observation drawn from a particular distribution. Therefore, on average, the vast majority of the 256 Gaussians contribute minimally to the likelihood of the data given a particular mixture model.

When evaluating mixture densities with pruned models, one can either treat these mixture weights as having a small but non-negligible value, or set them to zero[3]. Note that the mixture weights are renormalized in both cases, and thus the former is more or less equivalent to add-one smoothing. The latter can be thought of as exactly the opposite of smoothing - "roughening" the distribution. To investigate this, we set all but the top 16 values in each mixture weight distribution to zero and ran a number of trials on a K8-based workstation, varing the

[3]Meaning a very small number, since they are stored in log domain.

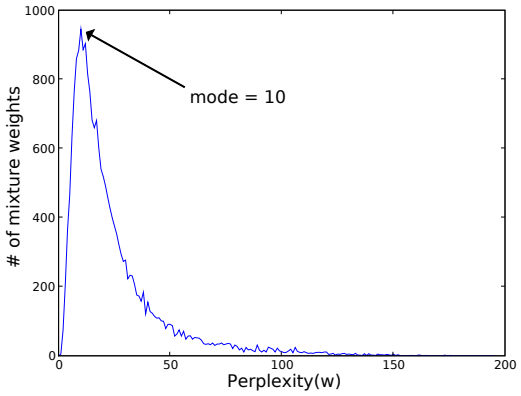

Figure 1: Perplexity distribution of mixture weights

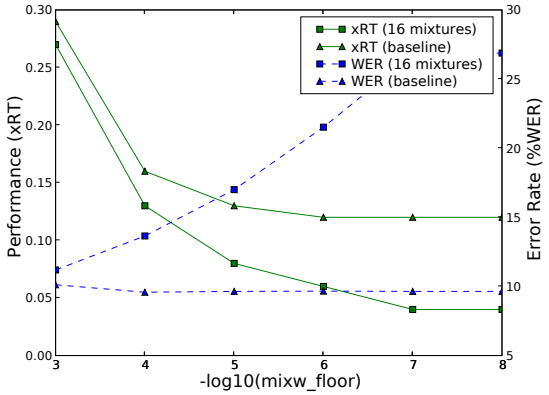

Figure 2: Smoothing vs. Roughening, 16 mixtures

mixture weight floor to produce either a smoothing or roughening effect. We discovered something initially surprising: "roughening" the mixture weights speeds up decoding significantly, while smoothing them slows it down. A plot of speed and error rate versus mixture weight floor is shown in Figure 2.

However, there is a simple explanation for this. At each frame, only the top N Gaussian densities are actually used to calculate the likelihood of the data:

$$p(x|\lambda_i) = \sum_{k \in topN} w_{ik} N(x; \vec{\mu}_{ik}, \vec{\sigma}^2_{ik})$$

When we remove mixture weights, we increase the probability that these top N densities will be matched with pruned-out weights. If we smooth the weights, we may raise some of these weights above their maximum-likelihood estimate, thus increasing

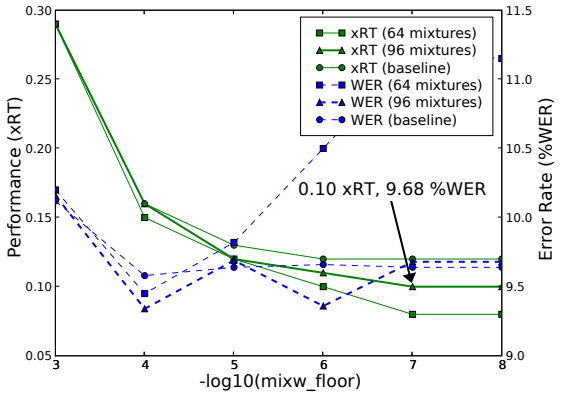

Figure 3: Speed-accuracy tradeoff with pruned mixtures

the likelihood for models whose top mixture weights do not overlap with the top N densities. This may prevent HMM states whose output distributions are modeled by said models from being pruned by beam search, therefore slowing down the decoder. By "roughening" the weights, we decrease the likelihood of the data for these models, and hence make them more likely to be pruned, speeding up the decoder and increasing the error rate. This is a kind of "soft" GMM selection, where instead of excluding some models, we simply make some more likely and others less likely.

As we increase the number of retained mixture weights, we can achieve an optimal tradeoff between speed and accuracy, as shown in Figure 3. Additionally, the perplexity calculation suggests a principled way to vary the number of retained mixture weights for each model. We propose setting a target number of mixture weights, then calculating a scaling factor based on the ratio of this target to the average perplexity of all models:

$$topK_i = \frac{target}{\frac{1}{N}\sum_{i=0}^{N} pplx(w_i)} pplx(w_i)$$

One problem is that many models have very low perplexity, such that we end up retaining only a few mixture weights. When the mixture weights are "roughened", this guarantees that these models will score poorly, regardless of the data. We compensate for this by keeping a minimum number of mixture weights regardless of the perplexity. Using a target of 96 mixtures, a minimum of 16, and a mixture weight floor of 10^{-8}, we achieve 9.90% word error rate in 0.09 times real-time, a 21% speedup with a 2.7% relative increase in error (baseline error rate is 9.64% on the desktop).

Using the same entropy-pruned mixture weights on the N800, we achieve an error rate of 9.79%, running in 1.19 times real-time, a 15% speedup with a 3.4% relative increase in error. After applying absolute pruning thresholds of 800 HMMs per frame and 5 words per frame, we obtained a 10.01% word error rate in 1.01 times real-time.

5 Conclusion

We have shown that a simple pruning technique allows acoustic models trained for large-vocabulary continuous speech recognition to be "scaled down" to run in real-time on a mobile device without major increases in error. In related work, we are experimenting with bottom-up clustering techniques on the mixture weights to produce truly scalable acoustic models, and subvector clustering to derive semi-continuous models automatically from well-trained fully-continuous models.

Acknowledgments

We wish to thank Nokia for donating the N800 tablet used in these experiments.

References

X. D. Huang. 1989. *Semi-continuous Hidden Markov Models for Speech Recognition*. Ph.D. thesis, University of Edinburgh.

D. Huggins-Daines, M. Kumar, A. Chan, A. Black, M. Ravishankar, and A. Rudnicky. 2006. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In *Proceedings of ICASSP 2006*, Toulouse, France.

Q. Huo and C. Chan. 1995. On-line Bayes adaptation of SCHMM parameters for speech recognition. In *Proceedings of ICASSP 1995*, Detroit, USA.

J. Leppänen and I. Kiss. 2005. Comparison of low footprint acoustic modeling techniques for embedded ASR systems. In *Proceedings of Interspeech 2005*, Lisbon, Portugal.

D. Paul and J. Baker. 1992. The design for the Wall Street Journal based CSR corpus. In *Proceedings of the ACL workshop on Speech and Natural Language*.

M. Woszczyna. 1998. *Fast Speaker Independent Large Vocabulary Continuous Speech Recognition*. Ph.D. thesis, University of Karlsruhe.