TEXT-INDEPENDENT SPEAKER VERIFICATION IN EMBEDDED ENVIRONMENTS

Bořivoj Tydlitát[1], Jiří Navrátil[2], Jason W. Pelecanos[2] and Ganesh N. Ramaswamy[2]

[1]Voice Technologies and Systems, IBM Czech Republic, V Parku 2294/4, 14800 Praha 4, Czechia
[2]IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA
e-mail: borivoj_tydlitat@cz.ibm.com, {jiri,jwpeleca,ganeshr}@us.ibm.com

ABSTRACT

A low-resource, text-independent speaker verification (SV) system with an efficient voice model compression technique is described, including its implementation exclusively with integer arithmetic. We describe and discuss the individual algorithmic steps, the integer implementation issues and its error analysis. Performance of the system is evaluated on data collected with iPaq devices and the impact of model compression as well as integer approximation on the accuracy is discussed.

Index Terms— Speaker verification, embedded systems, voice model compression, integer arithmetic

1. INTRODUCTION

Recent progress in speaker verification technologies was largely focused on server-based applications for telephony [1]. With current rapid development in built-in, mobile and portable systems and applications, it becomes feasible to deploy speech applications, including nontrivial ones, in these computing environments [2]. This, together with growing importance of all aspects of IT security, makes the deployment of speaker verification (SV) technology outside its traditional domain an attractive goal.

An "embedded system" can be for example a PDA, smartphone, car computer or a built-in controller in an instrument or appliance. Most often it uses resource-constrained, low-cost hardware, often with focus on low power, as many embedded systems are portable and battery-operated. A typical modern embedded system capable of speech processing uses a general-purpose 32-bit RISC processor. Examples of such architectures are ARM, IBM PowerPC and Hitachi SH-4.

SV technology in embedded systems faces several issues and challenges as follows:

- While the acoustic channel variability may be limited (single microphone per device), the system still may be used in diverse acoustic environments.
- The voice model may need to be stored on a smartcard or other secure medium of considerably limited capacity. Therefore its size becomes a significant factor.
- Computing power and memory may be limited by the architecture, hardware sizing and low-power requirements. For example, most embedded processors have no floating point arithmetic unit.

This paper presents a text-independent SV system that aims at achieving high accuracy while addressing the above challenges arising in embedded applications.

This work was partly supported by SAFIR project, funded by European Union (FP6-IP/e-Government, IST-507427).

2. ALGORITHMS

The speech is parameterized as a sequence of 19-dimensional Mel-Frequency Cepstral Coefficients (MFCC) and their first derivatives, post-processed via a marginal Gaussianization technique proposed in [3]. For modelling, we use a system based on Gaussian Mixture Models (GMM) and a likelihood-ratio detector. Each target model is a Maximum A-Posteriori (MAP) adaptation of a Universal Background model (UBM) in terms of its mean vectors, as proposed in [4]. More specifics on the training procedure are given in [5]. The scores in our system are calculated as a lower bound on the average log likelihood-ratio (LLR) of an utterance $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$ given a target and the UBM:

$$\frac{1}{T}\sum_t \log \frac{\sum_i p_{i2} p_2(\boldsymbol{x}_t|i)}{\sum_i p_{i1} p_1(\boldsymbol{x}_t|i)} \geq \frac{1}{T}\sum_{t,i} \gamma_{ti} \log \frac{p_{i2} p_2(\boldsymbol{x}_t|i)}{p_{i1} p_1(\boldsymbol{x}_t|i)} \quad (1)$$

with $\gamma_{ti} = P_1(i|\boldsymbol{x}_t) = p_1(\boldsymbol{x}_t, i)/\sum_j p_1(\boldsymbol{x}_t, j)$, and T the vector count (index 1 pertains to the UBM, index 2 to the target model) [6]. This *average componentwise* ratio, $\bar{l}$ typically compares in performance to the original LLR, but since in our system only the maximum scoring UBM component for a frame $\boldsymbol{x}_t$ is used, the posterior γ becomes an indicator function and the above bound becomes an equality. Due to the fact that the target GMM shares the precision matrix with the UBM, the Eq. (1) is a linear function of the feature space:

$$\begin{aligned} l(\boldsymbol{x}_t|M) &= \sum_i \gamma_{ti}\{\boldsymbol{a}_{1i}^T \boldsymbol{x}_t + a_{0i}\} \rightarrow \boldsymbol{a}_{1k}^T \boldsymbol{x}_t + a_{0k} \\ \boldsymbol{a}_{1i} &= (\boldsymbol{\mu}_{2i} - \boldsymbol{\mu}_{1i})^T \boldsymbol{\Sigma}_i^{-1} \\ a_{0i} &= \frac{1}{2}\boldsymbol{\mu}_{1i}^T \boldsymbol{\Sigma}_i^{-1} \boldsymbol{\mu}_{1i} - \frac{1}{2}\boldsymbol{\mu}_{2i}^T \boldsymbol{\Sigma}_i^{-1} \boldsymbol{\mu}_{2i} \end{aligned} \quad (2)$$

where $\boldsymbol{\mu}_{1i}, \boldsymbol{\mu}_{2i}$ are the UBM and the target mean vectors for the i-th component, respectively, $\boldsymbol{\Sigma}_i^{-1}$ is the corresponding UBM precision matrix, and k is the top scoring component. The linear coefficient vector $\boldsymbol{a}_1$ and the scalar a_0 are therefore a suitable set to represent the target model and offer a basis for compression as described below. The final scores are normalized using T-Norm technique [7].

3. MODEL COMPRESSION

For embedded systems in particular, it is essential to exploit speaker models with a small footprint and scoring schemes that are highly efficient. Such algorithms should sustain little to no degradation in recognition performance. In the optimized system[1] described in this paper only a small degradation in performance was observed.

[1]The efforts of Ming Liu are acknowledged for the work performed on model compression.

Parameter Type	Storage Bits (per dimension)	Number of Quantized Levels
$\boldsymbol{a}_1$	4	16
a_0	10	1024
soft counts	6	64

Table 1. Bit assignments for voice model compression.

3.1. A Compact Model Representation

Our approach to model compression involves storing a compact representation of the linear and bias parameters. This method is based on previous work [8], with the difference that we perform scalar, per-dimension compression, rather than vector compression to achieve a small model footprint.

This is achieved by compressing the value a_0 and each element in vector $\boldsymbol{a}_1$ from Eq. (2) for all Gaussian components. These compressed parameters are stored in the voice model and can be used for scoring (speaker verification). For further voice model adaptation, soft counts derived from the GMM MAP adaptation training are required as well.

The data items are compressed by nonlinear quantization. To minimize the mean square quantization error, the specific quantization tables (one for a_0, one for each dimension of $\boldsymbol{a}_1$ and one for soft counts) are based on the actual statistical distribution of its respective elements across all Gaussian components. These tables are part of the UBM data and are shared by all voice models. The bit resolution used for each item is listed in Table 1.

If each uncompressed parameter of a model is stored using 4-byte float precision for a 38-dimensional feature space, then the storage required relating to a single Gaussian component is 154 bytes. In contrast, the compressed model uses 21 bytes per mixture component. Thus, the compressed model achieves more than a 7 times reduction in size with only a small degradation in accuracy.

3.2. Rapid Enrollment and Scoring

Typically for GMM based speaker verification systems, speaker enrollment and scoring involves the probability density estimation of each speech frame given each Gaussian component within a GMM. This evaluation can involve significant computation time and a number of proposals have examined this with possible optimizations [9]. Another possibility is to reduce overhead by exploiting Gaussian prediction, or more specifically, where given the most significant scoring Gaussian component for the previous frame, predict the N most likely Gaussian components for the current speech frame and evaluate only those.

The prediction of the most likely Gaussian components may be achieved in multiple ways. One approach is the Kullback-Leibler distance criterion whereby the N-closest Gaussians are selected as possible generators of the frame in question. A more successful approach is based on the analysis of out-of-set audio data to infer the probabilistic transitions between Gaussians.

To determine the probabilistic transitions, for each speech frame (extracted from development data), the top few scoring Gaussians were noted. In this work it was assumed that the top 5 Gaussians contributed to the likelihood of their corresponding frame. A transition probability matrix may then be established. For example; given the top Gaussian index for a speech frame, a search table recording the top 512 most likely components (as used in this evaluation) for a 2048 component GMM may be constructed. Each Gaussian component of the UBM correspondingly contains a 512 component search table. Thus, when this information is incorporated for enrollment and scoring, a speed-up of almost four times can be achieved with minimal performance degradation. As a safety measure for avoiding dead-end transitions, a full Gaussian *refresh* search is performed once every 100 frames.

4. INTEGER ARITHMETIC

As already mentioned, CPUs in most embedded systems do not have a floating-point arithmetic unit and therefore they can perform arithmetic on real numbers, transcendental function calculation or even integer division only through library calls, whose cost typically is dozens of CPU instructions per operation. Therefore computationally intensive algorithms need to be implemented solely or mostly using native integer arithmetic.

4.1. Implementation

We have used two approaches to integerization of the SV algorithm. Feature calculation up to log-spectrum uses per-frame block floating point, i.e. integer math, where the power-of-two data scale is dynamic, but shared by all items of a vector [10]. The remaining code is implemented as fixed point using multiple static scaling factors chosen to completely prevent arithmetic overflows, as overflow detection and handling is expensive in high-level languages. The native 32-bit integer word is then used for intermediate results, and data values (variables) are represented with 12–15 bits (this is dictated primarily by need to avoid overflow on multiplication). This technique offers an integer algorithm implementation, whose cost in executed instructions is only slightly higher than in the original floating-point case. The only additional operations are the ones needed for fixed-point rescaling, usually after multiplication or long summation.

Math functions used at runtime (square root, exponential and logarithm) were implemented via table lookup with linear interpolation over a limited interval (e.g. $\langle 0, 1)$ or $\langle 1, 2)$, using their properties for rescaling values for arguments outside that range. This provided efficient and portable code and a reasonable, easy to tune tradeoff between data size and computing cost. For example, the implementation of square root has relative accuracy around $2^{-13} \approx 10^{-4}$. It uses circa 10 arithmetic operations and a 1 kB data table.

4.2. Error Analysis

With limited word-length arithmetic, the question of numerical stability and accuracy arises. In the following we provide some theoretical insight using elements of statistical error analysis.

First, let's consider a simple case of quantization or roundoff error. Let v be a real value represented in binary fixed point, i.e. by an integer quantity $Q(v)$ such that $Q(v) = \mathrm{round}\,(vS) = \left\lfloor vS + \frac{1}{2} \right\rfloor$, where the scale $S = 2^N$. The signed integer N can be interpreted as the position of the fixed binary point within $Q(v)$, i.e. v is approximated by $v^* = Q(v)2^{-N}$. The roundoff error $\varepsilon(v) = v^* - v$ can be considered a random variable with uniform distribution $U(-2^{-(N+1)}, 2^{-(N+1)})$ and its variance is $\mathrm{var}(\varepsilon(v)) = 2^{-2N}/12$.

It is worth noting that while every operation has potential roundoff error in floating point math, sums and products are accurate in fixed point arithmetic. The most common operation with roundoff error is downscaling (i.e. moving binary point right) that follows most multiplications or multiply-accumulate expressions.

In a signal processing algorithm like SV feature calculation, scoring or training, roundoff errors propagate and also new ones are introduced. To provide a statistical view of error propagation, let us

consider a single algorithmic step represented by an arbitrary vector function $\boldsymbol{y} = F(\boldsymbol{x})$, where $\boldsymbol{x} = (x_1 \ \ldots \ x_n)$ and $\boldsymbol{y} = (y_1 \ \ldots \ y_p)$. $F(\boldsymbol{x})$ could be, for example, the Fourier transform of a single audio frame or cepstra calculation. If $\boldsymbol{x}$ is a random variable with mean $\boldsymbol{x}_0$ and a covariance matrix $\boldsymbol{C}_x$, whose elements are sufficiently small, the output covariance matrix $\boldsymbol{C}_y$ can be approximated using a first-order differential as follows:

$$\boldsymbol{C}_y \approx \boldsymbol{F}_x(\bar{\boldsymbol{x}})\boldsymbol{C}_x\boldsymbol{F}_x^T(\bar{\boldsymbol{x}}) \tag{3}$$

where $\boldsymbol{F}_x$ is the Jacobian of F (see e.g. [11]). If we interpret $\boldsymbol{x}$ as a representing the accurate quantity $\boldsymbol{x}_0 = \bar{\boldsymbol{x}}$, with random error characterized by $\boldsymbol{C}_x$, then $\boldsymbol{C}_y$ is the covariance matrix of the error of $\boldsymbol{y}_0 = F(\boldsymbol{x}_0)$. The caveat of this formula is that it applies only to differentiable $F(\boldsymbol{x})$ and that the expression quickly becomes intractable even for a small number of algorithmic steps. Moreover, this type of analysis does not take into account the roundoff / approximation error of the calculation itself.

In practice, the elements x_i's (and y_j's) often can be considered statistically independent, $\boldsymbol{C}_x$ and $\boldsymbol{C}_y$ then are diagonal. The elements of $\boldsymbol{C}_y$ can therefore be considered separately and the equation (3) can be simplified to classical error propagation law ($f = F_j$ here):

$$\mathrm{var}(f(\boldsymbol{x})) = \sum_{i=1}^{n} \left(\frac{\partial f}{\partial x_i}\right)^2 \mathrm{var}(x_i) \tag{4}$$

Often the calculation / approximation $f^*(\boldsymbol{x})$ of $f(\boldsymbol{x})$ itself is loaded with roundoff or similar additive error $\varepsilon(f(\boldsymbol{x})) = f^*(\boldsymbol{x}) - f(\boldsymbol{x})$. Of course, the roundoff error is not really a random value, but a complicated and quickly changing function of the input vector $\boldsymbol{x}$ and can be treated as random and uncorrelated to $f(\boldsymbol{x})$ only when the values $\boldsymbol{x}$ and $f^*(\boldsymbol{x})$ themselves can be considered random with sufficiently "wide" distribution.

The variance $R_f(\boldsymbol{x}) = \mathrm{var}(\varepsilon(f(\boldsymbol{x})))$ of such error must be then added to (4), which becomes

$$\mathrm{var}(f^*(\boldsymbol{x})) = R_f(\boldsymbol{x}) + \mathrm{var}(f(\boldsymbol{x})) \tag{5}$$

Another aspect of roundoff errors are the rounded values of constants used in the calculation, e.g. the coefficients of the preemphasis filter, FFT sine/cosine tables, UBM values etc. The error they introduce is not random, but systematic and therefore cannot be analyzed using the above assumptions.

We performed error analysis by comparing the algorithm's intermediate results using both integer and float calculations and obtaining the statistics of cumulative absolute error $\varepsilon_I(y) = y_{\mathrm{int}} - y_{\mathrm{float}}$. As different intermediate results have different dynamics, we characterize their inaccuracy by relative cumulative roundoff error $\delta_I(y) = \varepsilon_I(y)/\sigma(y)$, where $\sigma(y)$ is the standard deviation of y.

Similarly, we have explored the effect of error from constant rounding by calculating the statistics of $\varepsilon_C(y) = y_{\mathrm{rnd}} - y_{\mathrm{float}}$. Here, y_{float}, y_{int} are values from the float, resp. integer variant of the algorithm. y_{rnd} come from a special variant of float algorithm, that uses all real constants rounded as they would be in their fixed-point form in the integer algorithm.

The statistics for selected intermediate values from model training process that characterize our SV implementation are summarized in Table 2. There are several interesting aspects of the data:

- The cumulative errors from rounded constants are smaller than total roundoff error at least by an order of magnitude and therefore can be considered insignificant.
- Overall, the integer feature extraction process has a good cumulative accuracy (0.1% relative for cepstra).

Value y	Dynamic of y $\sigma(y)$	Abs.Err. (total) $\sigma(\varepsilon_I(y))$	Abs.Err. (constants) $\sigma(\varepsilon_C(y))$	Rel.Err. (total) $\delta_I(y)$
Feature calculation:				
FFT	1,172	6.5	0.25	0.38%
Mel filters	19,200	24	2.7	0.13%
log filters	10.6	0.082	5×10^{-4}	0.77%
cepstra	66	0.37	0.05	0.10%
Training statistics:				
counts	6.4	0.26	0.05	3.3%
sums	6	0.19	5×10^{-5}	3.1%
Voice model data – see Eq. (2):				
a_0	0.67	0.054	2×10^{-4}	8%
$\boldsymbol{a}_1$	0.34	0.048	6×10^{-4}	14%

Table 2. Roundoff errors in model training.

- The sharp increase in the relative error in model statistics indicates that the results of searching the UBM for maximum-likelihood prototypes are very sensitive to small perturbations in the data.

5. DATABASE

The embedded audio database used for testing is part of a larger *in-house*[2] data collection spanning the telephony and embedded audio spaces. The goal of each data collection was to be able to focus on the separation of various speaker, channel and speech content artifacts. A journal of records described each audio sample accordingly. This form of collection was acquired in such a manner to facilitate experiments that would effectively measure effects of matched audio content spoken by impostors and mismatched audio content spoken by uninformed target speakers.

The embedded speech data set consists of multiple sessions from a collection of speakers recorded at 22,050Hz with 16-bit resolution on a HP$^{\mathrm{TM}}$ Pocket PC iPaq H4350 device. Each session consists of read and spontaneous answers to 30 prompts. It took about 5 minutes to record the session, yielding about 2 minutes of speech. Typically, the duration of each response was 2 to 20 seconds. An important note is that speakers were not permitted to provide another recording session on the same day.

The evaluation protocol utilizing this data was carefully structured to provide mutually exclusive speaker sets for system development and finally for system evaluation. System development consisted of training a Universal Background Model (UBM) trained from 535 sets of 30-item sessions spoken by 284 speakers. T-Norm speaker models were also trained from this set. The speaker recognition evaluation phase was performed using 178 models with each model trained on all 30 items for a speaker during their first recording session. For speaker testing, there were 903 test files from 181 speakers for a total of 159831 trials. Of the 181 test speakers, 4 were not from the enrollment speaker set. Each test utterance consisted or either one read or spontaneous item from the speaker's second recording session. The selected utterances typically ranged in duration from 2 to 10 seconds. The result is a configuration that provides a good coverage of spoken context.

[2]The authors acknowledge the contributions of William J. Ablondi, Allen Delmar and John Dildine for managing the embedded device audio data collection.

System	model compression	EER % T-Norm yes	T-Norm no
float	yes	1.6	2.9
	no	1.4	2.8
integer	yes	1.5	3.1
	no	1.3	2.8
float + noise	yes	1.4	2.9

Table 3. Scoring accuracy – float vs. integer system.

	Math	Avg. Audio Length [s]	% Real Time[1]	CPU[2] [Mcyc/s]
training	integer	167	18%	37
	float		400%	830
scoring	integer	8	27%	56
	float		410%	840

[1]CPU time per sec. of audio (22 kHz). [2]CPU cycles per sec. of audio.

Table 4. Integer system performance on iPaq 3600.

6. RESULTS

6.1. Accuracy

The system accuracy with a UBM consisting of 1024 components, measured in terms of the Equal Error Rate (EER), is shown in Table 3. Both training of the models and scoring was performed using the same arithmetic variant. Note the impact of T-Norm and the somewhat surprising improvement in EER for the integer system with T-Norm over the floating-point one. As this improvement has been seen in earlier experiments as well and shown to be statistically significant, we conducted an experiment using the float algorithm in which the integer roundoff error in feature calculation during training was replaced by random Gaussian noise, whose variance matched the variance of individual cepstral coefficients' roundoff error. The hypothesis was that such a training results in a model with a better coverage of voice variations of the same speaker. While the experiment did reproduce the accuracy improvement, other similar tests with varied noise level were inconclusive.

Further, as expected, a slight degradation in accuracy can be seen due to model compression.

6.2. CPU Performance and Resources

Performance of the integer SV training and scoring was measured on a real-world embedded system – iPaq 3600 handheld computer, running Linux. The device has Intel StrongARM 1110 processor (ARM v.4 architecture, 32-bit RISC, no FPU) with 206 MHz clock and slow memory (92 CPU cycles per 32 B cache line load). The average timing results are in Table 4. Clearly, the integer SV code consumes only a fraction of the device's CPU power, if real-time audio feed is considered. The float code is 14–22× slower.

The sizes of SV data are listed in Table 5. The sizes of UBM and voice models scale with the feature vector dimension and number of components, as evident from Section 3.

Data Item	Size	Note
UBM	836 kB	1024 prototypes
voice model (compressed)	21 kB	
T-Norm cohort	3747 kB	178 voice models

Table 5. Sizes of SV data.

6.3. Discussion

The described SV system provides accuracy high enough to be practically useful in embedded applications.

Compared to the telephony environments, the channel variability factor is considerably reduced in our experiments. Interestingly, the improvement in accuracy brought about by T-Norm (supposed to reduce the effect of the channel variability) is still significant.

Voice model compression provides size reduction, which is crucial in two ways. First, it is necessary for storing such models on secure media like smartcards. Second, the cohort size is reduced, enabling the use of the T-Norm even on devices with limited memory. The accuracy loss associated with the model compression is acceptable.

The integer implementation of SV fits well within the constraints of many current embedded systems. Its accuracy can paradoxically be even better than that of the original algorithm. There is anecdotal evidence of useful role that the roundoff noise may play in the training process, but more work is needed to further explore this phenomenon.

7. REFERENCES

[1] NIST Speaker Recognition Evaluations website, URL: http://www.nist.gov/speech/tests/spk/index.htm .

[2] T. Beran, V Bergl, R. Hampl, K. Krbec, J. Šedivý, B. Tydlitát, and J. Vopička, "Embedded ViaVoice," chapter Lecture Notes in Artificial Intelligence LNCS/LNAI 3206, also in Proc. of the 7th Int. Conf. on Text, Speech and Dialogue—TSD 2004, pp. 269–274, Springer-Verlag, 2004, ISBN 3-540-23049-1.

[3] J. Pelecanos and S. Sridharan, "Feature warping for robust speaker verification," in *Proc. Speaker Odyssey 2001*, Crete, Greece, June 2001.

[4] D.A Reynolds, T.F. Quatieri, and R.B. Dunn, "Speaker verification using adapted gaussian mixture models," *Digital Signal Processing*, vol. 10, no. 1-3, pp. 19–41, January/April/July 2000.

[5] G.N. Ramaswamy, J. Navrátil, U.V. Chaudhari, and R. Zilca, "The IBM system for the NIST 2002 cellular speaker verification evaluation," in *Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Hong Kong, April 2003, IEEE.

[6] J. Navrátil and G.N. Ramaswamy, "DETAC - a discriminative criterion for speaker verification," in *Proc. of the International Conference on Spoken Language Processing (ICSLP)*, Denver, CO, September 2002.

[7] R. Auckenthaler, M. Carey, and H. Lloyd-Thomas, "Score normalization for text-independent speaker verification systems," *Digital Signal Processing*, vol. 10, no. 1-3, pp. 42–54, January/April/July 2000.

[8] M. Novák, R.A. Gopinath, and J. Šedivý, "Hierarchical labeler algorithm for gaussian likelihoods computation in resource constrained speech recognition systems," URL: http://www.research.ibm.com/people/r/rameshg/novak-icassp2002.ps, 2002.

[9] J. McLaughlin, D. Reynolds, and T. Gleason, "A study of computation speed-ups of the GMM-UBM speaker recognition system," in *Proc. of the European Conference on Speech Communication and Technology (EUROSPEECH)*, 1999, vol. 3, pp. 1215–1218.

[10] A. Oppenheim, "Realization of digital filters using block-floating-point arithmetic," *IEEE Trans. Audio and Electroacoustics*, vol. 18, no. 2, pp. 130–136, 1970.

[11] K.O. Arras, "An introduction to error propagation: Derivation, meaning and examples of equation $C_y = F_x C_x F_x^T$," Tech. Rep. EPFL-ASL-TR-98-01 R3, Autonomous Systems Lab, Institute of Robotic Systems, Swiss Federal Institute of Technology Lausanne (EPFL), 1998, URL: http://www.nada.kth.se/ kai-a/papers/arrasTR-9801-R3.pdf .