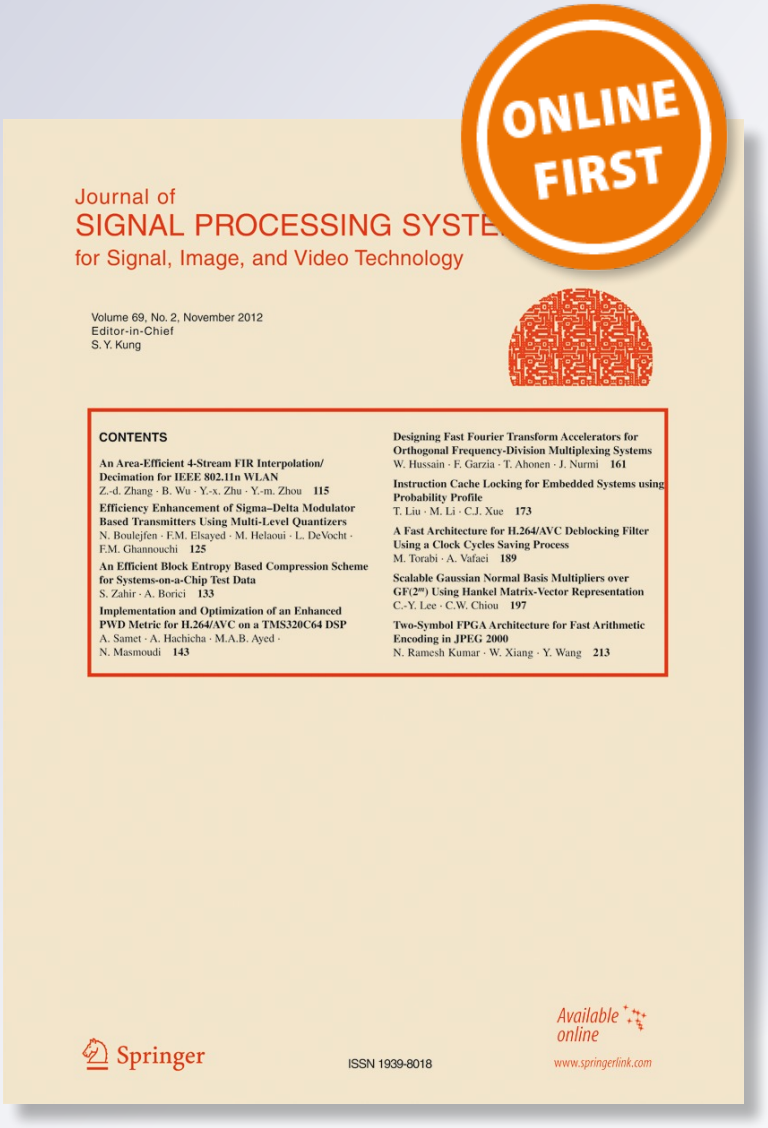
Fast Likelihood Computation in Speech Recognition using Matrices

Mrugesh R. Gajjar, T. V. Sreenivas & R. Govindarajan

Journal of Signal Processing Systems
for Signal, Image, and Video Technology
(formerly the Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology)

ISSN 1939-8018

J Sign Process Syst
DOI 10.1007/s11265-012-0704-4



Springer

Your article is protected by copyright and all rights are held exclusively by Springer Science +Business Media New York. This e-offprint is for personal use only and shall not be self-archived in electronic repositories. If you wish to self-archive your work, please use the accepted author's version for posting to your own website or your institution's repository. You may further deposit the accepted author's version on a funder's repository at a funder's request, provided it is not made publicly available until 12 months after publication.

Author's personal copy

J Sign Process Syst
DOI 10.1007/s11265-012-0704-4

Fast Likelihood Computation in Speech Recognition using Matrices

Mrugesh R. Gajjar · T. V. Sreenivas · R. Govindarajan

Received: 26 April 2012 / Revised: 29 August 2012 / Accepted: 13 September 2012
© Springer Science+Business Media New York 2012

Abstract Acoustic modeling using mixtures of multivariate Gaussians is the prevalent approach for many speech processing problems. Computing likelihoods against a large set of Gaussians is required as a part of many speech processing systems and it is the computationally dominant phase for Large Vocabulary Continuous Speech Recognition (LVCSR) systems. We express the likelihood computation as a multiplication of matrices representing augmented feature vectors and Gaussian parameters. The computational gain of this approach over traditional methods is by exploiting the structure of these matrices and efficient implementation of their multiplication. In particular, we explore direct low-rank approximation of the Gaussian parameter matrix and indirect derivation of low-rank factors of the Gaussian parameter matrix by optimum approximation of the likelihood matrix. We show that both the methods lead to similar speedups but the latter leads to far lesser impact on the recognition accuracy. Experiments on 1,138 work vocabulary RM1 task and 6,224 word vocabulary TIMIT task using Sphinx 3.7 system show that, for a typical case the matrix multiplication based approach leads to overall speedup of 46 % on RM1 task and 115 % for TIMIT task. Our low-rank approximation methods provide a way for trading off recognition accuracy for a further increase in computational performance extending overall speedups up to 61 % for RM1 and 119 % for TIMIT for an increase of word error rate (WER) from 3.2 to 3.5 % for RM1 and for no increase in WER for TIMIT. We also express pairwise Euclidean distance computation phase in Dynamic Time Warping (DTW) in terms of matrix multiplication leading to saving of approximately $\frac{1}{3}$ of computational operations. In our experiments using efficient implementation of matrix multiplication, this leads to a speedup of 5.6 in computing the pairwise Euclidean distances and overall speedup up to 3.25 for DTW.

Keywords Speech recognition · Acoustic likelihood computations · Low-rank matrix approximation · Euclidean distance matrix computation · Dynamic time warping

The major part of work was done when the first author was at Indian Institute of Science.

M. R. Gajjar (✉)
Siemens Corporate Research and Technologies,
Bangalore, 560100, India
e-mail: gajjar.mrugesh@gmail.com

T. V. Sreenivas
Department of Electrical Communication Engineering,
Indian Institute of Science, Bangalore, 560012, India
e-mail: tvsree@ece.iisc.ernet.in

R. Govindarajan
Supercomputer Education & Research Centre,
Indian Institute of Science, Bangalore, 560012, India
e-mail: govind@serc.iisc.ernet.in

1 Introduction

Acoustic likelihood computation is the most time consuming phase in most large vocabulary continuous speech recognition (LVCSR) systems, which takes up 30–70 % of the total recognition time [1–3]. Real-time

Published online: 02 October 2012

Author's personal copy

performance of an LVCSR task on general purpose processors has become a reality. However the application of speech recognition to a variety of systems such as hand held devices, require a much lower complexity of implementation, without sacrificing performance. Newer algorithms (e.g. fMPE [4]) for better speech recognition are even more compute intensive, which will bring down LVCSR performance even on general purpose processors, let alone power efficient handheld devices. There has been a continuous effort to speedup the acoustic likelihood computation performance in speech recognition systems [5–15].

We can classify the techniques proposed in the literature in to two broad categories. (1) algorithm centric (2) architecture centric. Algorithm centric techniques typically compute a subset of Gaussians [6], or a subset of Gaussian Mixture Models (GMM) [7], or a subset of feature vectors [5], or a subset of components within Gaussians [8] and approximate the rest from the likelihoods evaluated on the subset, based on some heuristic. The algorithm centric techniques also utilize the instantaneous knowledge of LVCSR search, e.g., GMMs that are outside the beam need not be evaluated; i.e., at each feature vector we may compute a subset of highest score Gaussians, and not all of them. To fully utilize the knowledge of the search part, algorithm centric techniques compute the likelihood of one feature vector at a time; it may require iterating through each component of the vector and stopping when certain condition is fulfilled. E.g., In Partial Distance Elimination (PDE) [9], likelihood computation is terminated for a GMM when the partial accumulated likelihood exceeds the total posterior value of current N-best candidate Gaussian. In [16], memory overhead problem of Vector Quantization (VQ) based Gaussian selection schemes (e.g., [6]) is addressed based on PDE [9] technique. In [17], a likelihood value for a Gaussian in previous speech frame is exploited to speed up its computation for the current frame. Because of the algorithm centric nature, these techniques would lead to complex memory reference patterns and branch instructions that are hard to predict [18]. Naturally, these program characteristics are not *architecture friendly*, as they incur extra stalls due to unavailability of operands or branch misprediction. Further, their ratio of arithmetic operations to memory references is low. This causes the speech recognition task to be memory intensive [1–3], and the memory system performance determines the speedup achieved. It is possible to devise efficient implementation of likelihood computation for a particular platform, leading to speedup due to better utilization of the hardware resources for the platform. E.g., in [19], acoustic likelihood computation is accelerated on Intel XScale based mobile CPU platform by utilizing architectural features such as SIMD (Single Instruction Multiple Data) arithmetic support [18] and deep pipelining. It uses techniques such as software pipelining [18] and acoustic model data partitioning to improve utilization of the program-managed on-chip memory. In [20], a field programmable gate array (FPGA) based LVCSR system is proposed that uses pipelined architecture to efficiently implement acoustic likelihood computations. Next, we describe approaches which are more general in nature in utilizing the architectural resources.

We can refer to the techniques in [10–12] as architecture centric. They build upon the components that are inherently more architecture friendly such as for blocked implementations of linear algebra operations [21]. The basic linear algebra operations, viz., dot product, matrix-vector multiplication and matrix-matrix multiplication, are architecture friendly since the blocked implementation of computations leads to improved cache hits and hence improves memory system performance. They do provide a nice interleaving of memory references and arithmetic operations. E.g., for a dot product, there are two memory loads and two arithmetic operations per component of the vectors. Since the task accomplished in signal processing algorithms, such as speech recognition depends on the number of arithmetic operations executed, it is important to increase the ratio of arithmetic operations (AOPS) to memory reference operations (MOPS) in an efficient implementation. The vector-vector product and matrix-vector product operations has this ratio as a constant. Matrix-matrix multiplication has a key characteristic of having the ratio of arithmetic operations to memory references which is linear in the size of the matrix instead of a constant. This also means that bigger matrix-matrix multiplication is more beneficial, overall. In other words, matrix-matrix multiplication leads to better reuse of memory data items and thus better cache memory efficiency.

The idea of speeding up the acoustic likelihood computation by expressing it in matrix-matrix multiplication form and implementing it with efficient Basic Linear Algebra Subroutines (BLAS) [21] was proposed in [10] for a real-time broadcast news transcription system. Later, [11–15] proposed to use hardware accelerator such as Graphics Processing Unit (GPU) to accelerate the matrix operations involved in likelihood computation. However, none of the methods in the literature are addressing the issue of modifying the algorithm to suit the architecture for efficient implementation. The algorithm centric methods will result in certain reduction in recognition performance, compared to a full search and full GMM computations.

Author's personal copy

In addition, these techniques do not exploit the current day parallel processing ability of General Purpose Processors (GPP) and GPUs. In the present work, we show that further computational benefit can be gained by low-rank approximation-factorization of the involved matrices along with the use of BLAS which are already optimized to the architectural features of GPP/GPU. We analyze the condition for such computational savings and the amount of savings that can be achieved. The matrix approximations do affect the Gaussian likelihoods and hence certain loss of recognition performance. We propose a novel technique to achieve the computational benefits with significantly lesser deterioration of recognition performance (Word Error Rate).

In Appendix A, we also show that the matrix multiplication based formulation can lead to saving approximately $\frac{1}{3}$ of computational operations in computing pairwise Euclidean distances between the Test and Template patterns in Dynamic Time Warping (DTW). In our experiments using efficient implementation of matrix multiplication, this leads to a speedup of 5.6 in computing the pairwise Euclidean distances and overall speedup up to 3.25 for DTW.

A technique for feature space dimensionality reduction namely Principle Component Analysis (PCA) [22, 23], aim to improve recognition performance by reducing spurious features. It can also can lead to similar computation reduction if implemented using the Matrix multiplication based approach. In Appendix B, we compare important differences of the PCA method with respect to our work.

The paper is organized by starting from the matrix-matrix multiplication for likelihood computations in speech recognition and applying the well-known blocking or tiling approach to improve its performance. Next, we recognize the potential for computational improvement in the matrix multiplication approach and approximate the Gaussian parameter matrix by a lower rank matrix, thus reducing the matrix multiplication computation to a fraction depending on the lower rank. Such direct low-rank approximation results in deterioration of recognition performance. Therefore we propose a feature-sensitive low rank approximation which is computationally efficient while keeping the error rate within control. The application of this in Sphinx 3.7 speech recognizer on a 1,138 word vocabulary RM1 task, has shown that, for a typical case, the matrix-matrix multiplication approach leads to overall speedup of 46 %. Using the feature-sensitive low rank approximation, it is further improved to 60 % for an increase in word error rate from 3.2 to 6.6 %. Recognizing that the word error rate is directly affected by errors in individual Gaussian likelihoods, minimizing errors in the Gaussian parameters is not the best way to improve word error rate. Hence, we approximate the likelihood matrix and use the training data to estimate an optimum low-rank Gaussian parameter matrix; This approximation increased the WER from 3.2 to 3.5 %. Thus gaining a computational speedup of 60 % with a small increase in WER.

Further we conduct experiments on 6,224 word vocabulary TIMIT task for the optimum low-rank approximation method. Due to small amount of training data available we could only successfully train acoustic models with 9,104 Gaussians and 1,138 tied states (Senones). Due to smaller acoustic model, the overall computational gain is limited but it leads to improvements in WER.

2 Acoustic Likelihood Computation Using Matrix Multiplication

The likelihood computation problem can be stated as follows: Given a set of m feature vectors and a set of n Gaussians specified by the speech recognition model, we compute log-likelihood for all pairs of feature vectors and Gaussians. For each pair, computing the likelihood can be reduced to a dot product operation between augmented vectors obtained from feature vector and Gaussian parameters. So, the entire likelihood computation problem can be converted to a matrix-matrix multiplication problem, the matrices being derived from feature vectors and Gaussian parameters.

2.1 Matrix Equation for Multivariate Gaussian

The pdf of multivariate Gaussian variable of dimension d with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ is:

$$f_X(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}\,|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{\mathbf{1}}{\mathbf{2}}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-\mathbf{1}}(\mathbf{x}-\boldsymbol{\mu})\right)$$

For each feature vector $\mathbf{x}$ of dimension d we calculate the negative log-likelihood of each Gaussian j, $1 \leq j \leq n$. The total number of Gaussians is n and these Gaussians are used by the system to model various states of Hidden Markov Models (HMM) in the speech recognition model; i.e., $\mathcal{N}(\mu_\mathbf{j}, \boldsymbol{\Sigma}_\mathbf{j})$.

Author's personal copy

If the covariance matrix $\mathbf{\Sigma_j}$ is assumed diagonal (as it is often the case), the individual likelihood computation can be expressed as:

$$-\log f_X(\mathbf{x}) = c_j + \frac{1}{2}\sum_{i=1}^{d}\frac{(x_i-\mu_{ji})^2}{\sigma_{ji}^2}$$
$$= c_j + \frac{1}{2}\sum_{i=1}^{d}\frac{x_i^2}{\sigma_{ji}^2} - \sum_{i=1}^{d}\frac{x_i\mu_{ji}}{\sigma_{ji}^2} + \frac{1}{2}\sum_{i=1}^{d}\frac{\mu_{ji}^2}{\sigma_{ji}^2}$$

where $1 \le j \le n$ and c_j is independent of $\mathbf{x}$.

Expanding the quadratic term $\frac{(x_i-\mu_{ji})^2}{\sigma_{ji}^2}$, (resulting due to the diagonal covariance matrix) we can see that the likelihood computation involves a summation of three different dot products: (1) vector $\mathbf{x^2}$ with vector $\frac{\mathbf{1}}{\sigma^2}$ (2) vector $\mathbf{x}$ with vector $\frac{-\mathbf{2}\mu}{\sigma^2}$ and (3) vector μ^2 with vector $\frac{\mathbf{1}}{\sigma^2}$. Thus it is a set of dot products between augmented vectors from the feature vector and the Gaussian parameter vectors, plus a Gaussian dependent constant addition. Now, let us consider the sequence of feature vectors in speech recognition, i.e., $\mathbf{x_1}$ to $\mathbf{x_m} \equiv \mathbf{X}$, an $m \times d$ matrix where each row is a feature vector. $\mathbf{X^2}$ is also an $m \times d$ matrix where each element is a square of corresponding element in $\mathbf{X}$. Let $\mathbf{M}$ be a $d \times n$ matrix of columns of $\frac{-2\mu_i}{\sigma_i^2}$, $1 \le i \le d$ and $\mathbf{S}$ be a $d \times n$ matrix of columns of $\frac{1}{\sigma_i^2}$, $1 \le i \le d$, then the matrix of likelihoods can be expressed as:

$$\mathbf{L_{tot}} = \exp\left(-\frac{\mathbf{1}}{\mathbf{2}}\left(\mathbf{X^2S} + \mathbf{XM} + \mathbf{C_2}\right)\right)\cdot\mathbf{C_1}$$

$\mathbf{C_1}$ is an $n \times n$ diagonal matrix comprising the scaling constants for each Gaussian and $\mathbf{C_2}$ is an $m \times n$ matrix of rank-1 comprising of identical rows.

The negative log-likelihood matrix can be expressed as:

$$-\mathbf{2}\log\mathbf{L_{tot}} = \mathbf{X^2S} + \mathbf{XM} + \mathbf{C_2} - \mathbf{2}\log\mathbf{C_1} \quad (1)$$

$\log\mathbf{C_1}$ is an $m \times n$ matrix of rank-1 with identical rows, similar to $\mathbf{C_2}$.

We can compute the log-likelihood efficiently by rewriting Eq. 1 as a single dot product:

$$-\mathbf{2}\log\mathbf{L_{tot}} = \left[\mathbf{X^2}|\mathbf{X}\right]\cdot\begin{bmatrix}\mathbf{S}\\ \mathbf{M}\end{bmatrix} + \mathbf{C} \equiv \mathbf{FG} + \mathbf{C} = \mathbf{L} + \mathbf{C} \quad (2)$$

where $\mathbf{F} \equiv [\mathbf{X^2}|\mathbf{X}]$ and $\mathbf{G} \equiv [\frac{\mathbf{S}}{\mathbf{M}}]$, obtained by concatenation of matrices along columns and rows respectively, are the augmented feature vector matrix and the Gaussian parameter matrix, respectively.

2.2 Low-Rank Factorization for Computation Reduction

Computing the negative log likelihood matrix consists of the $(m \times 2d) \times (2d \times n)$ matrix-matrix multiplication as shown by Eq. 2; The augmented Gaussian parameter matrix $\mathbf{G}$ is independent of input signal $\mathbf{X}$ and hence can be factorized into lower rank factors apriori.

The total number of arithmetic operations (multiplication and addition) to compute $\mathbf{FG}$ is $4mnd$. However, we examine reducing the rank of $\mathbf{G}$ by k, so that it is possible to find a factorization of $\mathbf{G} = \mathbf{G_1G_2}$, where $\mathbf{G_1}$ is of size $2d \times 2d-k$ and $\mathbf{G_2}$ is of size $2d-k \times n$, assuming $n >> 2d$. Also, let us assume that this factorization is exact and has been precomputed and saved. Thus, using the factorization, the total number of arithmetic operations for likelihood computation is given by

$$\mathbf{FG_1G_2} = ((\mathbf{FG_1})\mathbf{G_2});$$
$$\#operations = 4md(2d-k) + 2mn(2d-k)$$
$$= 8md^2 - 4mdk - 2mnk + 4mnd$$

Since the original $\mathbf{FG}$ matrix multiplication requires $4mnd$ operations, we can get a saving in arithmetic operations if

$$8md^2 - 4mdk - 2mnk < 0 \quad (3)$$

Let us consider $n = 2rd$, and $k = 2fd$, $r >> 1$, $0 \le f \le 1$. Since $n >> 2d$, this expresses the number of Gaussians as a multiple r of the common dimension $(2d)$ and the reduction in rank as a fraction of the common dimension (full rank). Using these in Eq. 3, the condition for savings in computation is given by:

$$f > \frac{1}{(r+1)} \quad (4)$$

The number of operations saved is given by

$$-(8md^2 - 4mdk - 2mnk) = 4mnd\left(f - \frac{1-f}{r}\right) \quad (5)$$

$4mnd$ being the original number of computations, the fraction of original operations saved is: $\left(f - \frac{1-f}{r}\right)$.

Thus the fraction of computations saved is less than the fraction of reduction in the rank f. However, this quantity approaches the fraction f as r increases to a large value; i.e., for a very large set of Gaussians, the fraction of reduction in arithmetic operations approaches the rank reduction factor f.

Figure 1 shows the fraction of arithmetic operations saved for different values of f against r. For $r \sim 100$ we approach the maximum limit. In speech recognition

Author's personal copy

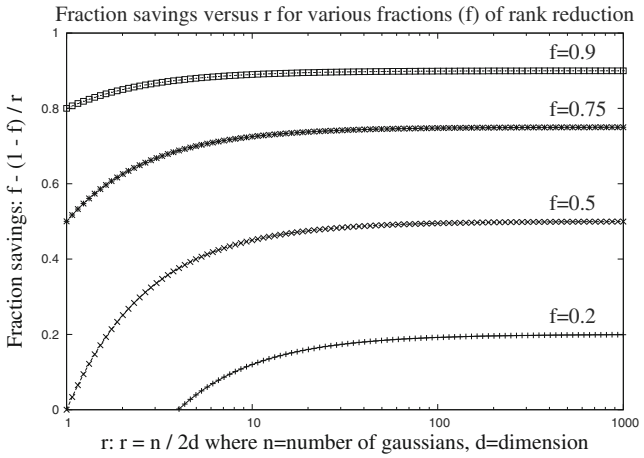


Figure 1 Fraction of arithmetic operations saved versus r.

applications, the number of Gaussians is two to three orders of magnitude larger than the dimension $d = 39$ (r between 50 to 500). Hence, we can expect the reduction in computation using factorization $\simeq f$.

3 Low-Rank Approximation of G

The factorization of $\mathbf{G} = \mathbf{G_1} \cdot \mathbf{G_2}$ can be either exact or approximate. If it is exact, there is no change in the likelihoods computed and we can expect the same speech recognition performance with a reduction in computation by a fraction $\left(f - \frac{1-f}{r}\right)$. If the factorization is inexact (approximate), then there would be some change in the recognition performance, along with the reduction in computation. We consider first inexact factorizations, since they are more promising.

An approximation of a matrix by one with rank deficiency of k can be obtained by replacing the smallest k singular values by zeroes in the Singular Value Decomposition (SVD) of the matrix. Such an approximation minimizes the Frobenius norm of the error matrix for a given k [24]. We use SVD based approximate factorization for $\mathbf{G} = [\frac{\mathbf{S}}{\mathbf{M}}]$ and $\mathbf{S}$ separately. Since, $\mathbf{S}$ is a non-negative matrix, Non-negative Matrix Factorization [25] can also be used which minimizes KL divergence between $\mathbf{S}$ and the approximated matrix. Thus, we compare the effects of rank reduction on $\mathbf{S}$ using two different factorization approaches, namely SVD and NMF on $\mathbf{S}$.

While the rank reduction reduces the number of arithmetic operations, it also introduces certain error in the likelihood computation. We measure the induced error in terms of the word error rate (WER) seen in the LVCSR.

3.1 Feature-Sensitive Low-Rank Approximation

Since it is clear that factorization error will cause an increase in speech recognition WER, we did an analysis of the error distribution in the approximated Gaussian parameter matrix $\mathbf{S}$ (using NMF, SVD) and $\mathbf{M}$ (using SVD). We find that the approximation errors are not distributed uniformly within $\mathbf{S}$ and $\mathbf{M}$. We find that errors are very high (50 to 100 %) in the rows corresponding to the initial 3–4 Mel-Frequency Cepstral Co-efficients (MFCC) components and also the corresponding 3–4 velocity and acceleration co-efficients. (The feature vectors are of dimension 39, consisting of 13 MFCC and corresponding velocity and acceleration co-efficients). The lower order MFCC co-efficients contain the spectral envelope of the speech signal which is considered the most important feature for recognizing speech [26]. As these lower order MFCC components are important for speech recognition, we propose a feature dependent approximation-factorization in which the sensitive rows in Gaussian parameter matrices are retained and the factorization is performed on the remaining rows. E.g., we retain the 9 components (first 3 of MFCC, velocity, and acceleration co-efficients) unchanged and approximate the remaining 30 dimensional matrix in $\mathbf{S}$ and $\mathbf{M}$. In a similar manner we retain 18 components in $\mathbf{G} = [\frac{\mathbf{S}}{\mathbf{M}}]$ and approximate the rest 60 rows. Using this method we can reduce the overall approximation error and hence control the WER.

4 Optimum Low-Rank Approximation of G

We consider deriving an optimum low-rank approximation $\hat{\mathbf{G}}$ through the use of likelihood matrix $\mathbf{L}$. The optimality here is to reduce the errors in likelihood instead of the errors in Gaussian parameters, considered earlier. Since our goal is to minimize the WER which is based on the likelihood, reduction of the errors in $\mathbf{L}$ should be better than reduction of errors in $\mathbf{G}$. However, this optimization will depend on $\mathbf{X}$, the input feature data and not independent as in the factorization of $\mathbf{G}$. We use the training data $\mathbf{X}$ which has been earlier used to derive $\mathbf{G}$, but this time to optimize computation. We could also optimize a measure of discriminability, instead of just the likelihood, constrained on the required amount of computation. However, in the present approximation we consider SVD of $\mathbf{L}$ and derive the optimum approximation for $\mathbf{G}$ so as to minimize the sum of squared errors in $\mathbf{L}$ using the augmented feature vector matrix $\mathbf{F}$. Let us denote $\tilde{\mathbf{G}}_{\mathbf{opt}} = \mathbf{svd_k}(\mathbf{G})$ as low-rank approximation of $\mathbf{G}$ to rank $2d - k$. Let $\mathbf{L} = \mathbf{FG}$ be the likelihood matrix as shown

Author's personal copy

in Eq. 2. Let $\tilde{\mathbf{L}} = \mathbf{F}\tilde{\mathbf{G}}_{\mathbf{opt}}$ be the approximated likelihood matrix resulting from the approximation of $\mathbf{G}$.

Note that, the recognition accuracy is affected directly by the errors $\mathbf{L} - \tilde{\mathbf{L}}$, in the likelihood matrix $\mathbf{L}$. The approximation error in the Gaussian parameter augmented matrix $\mathbf{G}$, gets modified through the data vector matrix $\mathbf{X}$. In Section 3, we approximated $\mathbf{G}$, since it is independent of the input data $\mathbf{X}$. However, we recognize that the low-rank approximation can be realized based on $\mathbf{L}$ instead of $\mathbf{G}$. The data dependency in $\mathbf{L}$ can be addressed by exploiting the training data itself for approximating $\mathbf{L}$. Thus,

$$\tilde{\mathbf{L}}_{\mathbf{opt}} = \mathbf{svd_k}(\mathbf{L}) = \mathbf{svd_k}(\mathbf{FG}) \tag{6}$$

Since we need an approximation $\hat{\mathbf{G}}$, applicable to any data $\mathbf{F}$, we can resort to pseudo-inverse, using the training data $\mathbf{F}$.

$$\hat{\mathbf{G}} = \mathbf{pInv}(\mathbf{F}) \cdot \tilde{\mathbf{L}}_{\mathbf{opt}} = \mathbf{pInv}(\mathbf{F}) \cdot \mathbf{svd_k}(\mathbf{FG}) \tag{7}$$

($\mathbf{pInv}(\mathbf{F})$ is pseudo inverse of $\mathbf{F}$.) The rank of $\tilde{\mathbf{L}}_{\mathbf{opt}}$ is $2d - k$ given $n > 2d$ and the rank of $\mathbf{pInv}(\mathbf{F})$ is $2d$ given $m > 2d$. Thus, the rank of $\hat{\mathbf{G}}$ will be $2d - k$ because in matrix multiplication rank of the resultant matrix is the minimum of the ranks of the multiplicands.

Here $\mathbf{F}$ can be thought of as the augmented feature matrix for $\mathbf{a}$ test utterance. In practice, we should find a single $\hat{\mathbf{G}}$ that should be good for all potential test utterances. Since we use training utterances as a model for test utterances we can find a single $\hat{\mathbf{G}}$ for the entire training set. Let $\mathbf{Y_i}$, $1 \leq i \leq N$ be the augmented feature matrices for the training set utterances. We would like to find $\hat{\mathbf{G}}^*$ that minimizes the sum of Frobenius norms of all error matrices corresponding to all utterances in the training set.

$$\hat{\mathbf{G}}^* = \mathbf{argmin}_{\hat{\mathbf{G}}} \left\{ \sum_{\mathbf{i=1}}^{\mathbf{N}} \left\| \mathbf{Y_i G} - \mathbf{Y_i}\hat{\mathbf{G}} \right\|_{\mathbf{F}} \right\} \tag{8}$$

where $\|\mathbf{A}\|_{\mathbf{F}}^{\mathbf{2}} = \sum_{\mathbf{i,j}} \mathbf{A}_{\mathbf{ij}}^{\mathbf{2}}$ is the Frobenius norm of $\mathbf{A}$. Since the squared Frobenius norm of an augmented matrix is the sum of squared Frobenius norms of its parts, Eq. 8 gives $\hat{\mathbf{G}}^* = \hat{\mathbf{G}}_{\mathbf{Y}}$, where $\hat{\mathbf{G}}_{\mathbf{Y}}$ is the optimum approximation obtained using all training data, putting $\mathbf{F} = \mathbf{Y} = \begin{bmatrix} Y_1 \\ \vdots \\ Y_N \end{bmatrix}$ in Eq. 7.

For large training sets $\mathbf{Y}$, computing Eq. 7 with $\mathbf{F} = \mathbf{Y}$ can be prohibitively expensive in terms of memory and CPU time. In experiments, we select a small subset of feature vectors randomly out of a large $\mathbf{Y}$ and use it as $\mathbf{F}$.

5 Experimental Results

We have conducted experiments using Sphinx 3.7 Continuous Speech Recognition (CSR) system running on a compute node with dual Intel E5440 Xeon 2.83 GHz processors. Each processor is a quadcore with 2 cores sharing 6 MB L2 cache. Thus, we have total 8 cores in the node with 16 GB RAM. All the computations are performed using single precision floating point operations. We report results using two speech databases namely RM1 and TIMIT.

5.1 Performance Results on RM1

RM1 database has vocabulary of 1,138 words. The RM1 test set consists of 600 utterances. The RM1 acoustic model consists of 1,935 GMMs with 8 Gaussians per mixture, thus a total of $n = 15{,}480$ Gaussians. Each feature vector is $d = 39$ dimension MFCC with velocity and acceleration co-efficients. On an average, a test utterance consists of $m = 340$ feature vectors. The training set $\mathbf{Y}$ consists of 1,600 utterances totalling 549,216 feature vectors and we select 700 feature vectors randomly out of it to denote as $\mathbf{F}$ for computing Eq. 7. We find that there is no significant change in recognition performance for larger $\mathbf{F}$.

5.1.1 Performance with Single Threaded BLAS

Table 1 shows average single threaded execution time based on five repetitions of the experiment, and the corresponding speedup factors along with the word error rate for the different schemes. We have used GotoBLAS 1.26 library for efficient linear algebra routines.

Baseline shows the performance without any modification to Sphinx 3.7. There are a total of 1,935 GMMs (*senones* in Sphinx terminology) trained with 8 Gaussians per mixture. This results in potentially 15,480 Gaussians to to evaluated against an incoming feature vector. To speed up the likelihood computation, Sphinx uses an algorithm centric technique named Context Independent senone based Gaussian Selection (CIGS) [5]. This technique results in computing 3,580 Gaussians per feature vector on an average, instead of the 15,480. Since the Gaussian selection algorithm is working in an iterative manner per feature vector, it is not convenient to use BLAS for the system.

BLAS scheme consists of using matrix multiplication to calculate all the likelihoods; it calculates *all* 15,480 Gaussians for all feature vectors in an utterance in one call. Even after evaluating roughly 4 times more number of Gaussians, the BLAS scheme results in

Author's personal copy

Table 1 Total single threaded execution time in seconds and recognition performance in word error rate for Sphinx 3.7 with 1,138 word vocabulary RM1 CSR task, Baseline=Sphinx 3.7 with CIGS.

Scheme	Eff. rank of $\left[\frac{S}{M}\right]$	Time	Speedup over BLAS	Speedup over Baseline	WER %
Baseline	78	123.15	0.69	1	3.2
BLAS	78	84.5	1	1.46	3.2
Part 1: Basic low rank approximation of S					
NMF 25	64	102.37	0.83	1.2	39
SVD 25	64	79.41	1.06	1.55	63
Part 2: Feature-sensitive low rank approximation of S					
FSVD-(9,16)	64	79.53	1.06	1.55	6.3
FSVD-(9,10)	58	78.2	1.08	1.57	6.6
FSVD-(9,8)	56	77.08	1.10	1.60	7.9
FSVD-(9,4)	52	76.12	1.11	1.62	8.0
FNMF-(9,16)	64	92.26	0.92	1.33	3.5
FNMF-(9,10)	58	82.2	1.03	1.5	4.9
FNMF-(9,8)	56	80.51	1.05	1.53	5.9
FNMF-(9,4)	52	77.11	1.10	1.6	6.6
Part 3: Feature-sensitive low rank approximation of $\left[\frac{S}{M}\right]$					
FSVD-(18,40)	58	82.21	1.03	1.5	4.1
FSVD-(18,34)	52	82.57	1.02	1.49	4.4
FSVD-(18,30)	48	82.54	1.02	1.49	5.8
FSVD-(18,20)	38	88.29	0.96	1.39	9.0
Part 4: Optimum low rank approximation of $\left[\frac{S}{M}\right]$ using training data					
SVD* 78	78	84.98	0.99	1.45	3.2
SVD* 72	72	82.39	1.03	1.49	3.4
SVD* 60	60	79.67	1.06	1.55	3.5
SVD* 54	54	78.58	1.08	1.57	3.5
SVD* 48	48	76.4	1.11	1.61	3.5
SVD* 38	38	76.28	1.11	1.61	4.0
SVD*-(19,19)	38	79.13	1.07	1.56	3.8

a speedup of 1.46 compared to the baseline version, which does not use BLAS but use CIGS. Note that there is no difference between **Baseline** and **BLAS** schemes with respect to recognition performance, since BLAS computes all the 15,480 Gaussians exactly without any approximation. Since we have not modified the network search part, the Viterbi search will proceed identically in both Baseline and BLAS schemes.

Part 1 of Table 1 shows the effect of using the data independent factorization of **S** through SVD and NMF. This factorization leads to larger error rates (39–63 %).

In the second part of the Table 1, we compare data independent factorization of **S** using both SVD and NMF. The schemes FNMF-(A,B) and FSVD-(A,B) denote low rank approximation of **S** with feature-sensitive A rows retained and B is the rank to which remaining (39-A) rows are reduced. As the effective rank increases, we see significant reduction in the word error rate (WER) and increase in the execution time. FNMF schemes have lower WER than the corresponding FSVD schemes. We suppose that WER is more sensitive to reduction in KL divergence than the Frobenius norm. We plan to study this effect as part of our future work. In Section 5.3, we explore why the speedup is only 10 % even after reducing the likelihood computations by one third (rank 52).

In the third part of the Table 1, the concatenated Gaussian parameter matrix $[\frac{\mathbf{S}}{\mathbf{M}}]$ was approximated as a single entity with rank (A,B) using SVD. As the rank decreases, the WER increases but the execution time does not reduce beyond some point. This is explained by the fact that the network search of LVCSR takes longer to compensate for approximations (see Section 5.3). However this effect is not visible in Part 2 where we factorize only **S**. We plan to explore this effect further in future work.

In the fourth part of the Table 1, the scheme SVD* N denotes optimum low rank-N approximation using all the training data. Here, the concatenated Gaussian parameter matrix $[\frac{\mathbf{S}}{\mathbf{M}}]$ is approximated as a single entity with rank N. E. g., SVD* 38 scheme reduces the rank of $[\frac{\mathbf{S}}{\mathbf{M}}]$ to 38 from 78 (approximately half). This is in contrast to the SVD*-(19,19) scheme which approximates the Gaussian parameter matrices **S** and **M** separately, with rank 19 each, and thus the effective rank is 38. It can be seen from part 4 of Table 1 that, as effective rank

Author's personal copy

increases, speedup over BLAS reduces and approaches 1; also WER approaches to the baseline WER 3.2 %. We get the best performance for SVD* 48 with mild increase in WER of 3.2 to 3.5 % and a speedup of 61 % over the baseline and 11 % speedup over BLAS. In the Section 5.3 we explore further the speedup factor 11 %, for a rank reduction by half (rank 38). Thus, we see that it is possible to trade off accuracy to computational gain.

Thus, both low rank approximations (FNMF and FSVD) result in a speedup of ∼60 % compared to Baseline (∼10 % compared to BLAS) using optimum low-rank approximation of the Gaussian parameter matrix. The training data based optimization results in only moderate increase in word error rate. More importantly, this approach enables trading off accuracy for a gain in performance.

5.1.2 Performance with Multi-Threaded BLAS

In Table 2, we report results with multi-threaded implementation of matrix-matrix multiplication using GotoBLAS2 v1.13 library. Multi-threaded BLAS allows us to utilize upto 8 cores available in the compute node.

Observations:

1. Speedups upto 36 % with 8 threads. Speedups do not increase linearly as number of threads increase.
2. No improvement in performance using multiple threads due to low-rank factorizations.
3. Performance degrades little for 4 and 8 number of threads as rank decreases.

5.1.3 Performance with Streaming Input

In performance results upto now, we evaluate Gaussian likelihoods for whole utterance **F** at once using matrix-matrix multiplication routine *sgemm*. This introduces a delay in real-time recognition process, because we need to wait till the utterance is spoken fully. In this section, we explore computing likelihoods one vector at a time using matrix-vector multiplication using *sgemv* routine of BLAS. Implementation using *sgemv* is slower compared to *sgemm*, because *sgemm* has higher AOPS/MOPS ratio. In Table 3, we report results with multi-threaded implementation of matrix-vector multiplication using GotoBLAS2 v1.13 library.

Table 2 Sphinx 3.7 with Multithreaded BLAS on RM1 task.

Scheme	1 thread	2 threads	4 threads	8 threads
Baseline	123.15			
BLAS	80.66	68.33	61.40	59.14
SVD *60	75.81	66.65	61.06	58.81
SVD *48	72.84	66.44	61.82	59.49
SVD *38	72.23	66.65	62.60	61.81

Table 3 Sphinx 3.7 with Multithreaded BLAS on RM1 task on streaming input.

Scheme	1 thread	2 threads	4 threads	8 threads
Baseline	123.15			
BLAS	208.02	103.50	82.79	67.88
SVD *72	171.31	96.65	79.04	69.16
SVD *60	131.30	91.77	75.29	67.60
SVD *54	126.53	91.55	75.52	67.65
SVD *48	112.93	88.22	73.93	67.10
SVD *38	109.28	89.41	75.49	69.09

Observations:

1. Single threaded performance of BLAS is significantly slower than Baseline by 69 %. We start to get performance improvements over Baseline at rank 48.
2. Unlike results in Table 2, performance improves significantly as rank goes lower for low-rank matrix approximations.
3. Speedups upto 3.06 for BLAS and upto 1.58 for SVD* 38 using 8-threads. Compared to results in Table 2, this shows that *sgemv* scales better than *sgemm* for our application.
4. No significant improvement in performance using multiple threads due to low-rank factorizations.

5.2 Performance Results on TIMIT Database

TIMIT database has vocabulary of 6,224 words. The TIMIT test set consists of 1,680 utterances. The RM1 acoustic model consists of 1,138 GMMs with 8 Gaussians per mixture, thus a total of $n = 9{,}104$ Gaussians. Each feature vector is $d = 39$ dimension MFCC with velocity and acceleration co-efficients. On an average, a test utterance consists of $m = 308$ feature vectors. The training set **Y** consists of 4,620 utterances totalling 1,414,504 feature vectors and we select 306 feature vectors randomly out of it to denote as **F** for computing Eq. 7. For all results with TIMIT database we use GotoBLAS2 v1.13 for efficient linear algebra operations.

5.2.1 Performance with Single Threaded BLAS

For TIMIT database we only evaluate the performance of our optimum low-rank matrix approximation method presented in Section 4. In Table 4, we report single threaded performance results of TIMIT speech database.

Author's personal copy

Table 4 Total single threaded execution time in seconds and recognition performance in word error rate for Sphinx 3.7 with 6,224 word vocabulary TIMIT CSR task, Baseline=Sphinx 3.7 with CIGS.

Scheme	Eff. rank of $[\frac{S}{M}]$	Time	Speedup over BLAS	Speedup over baseline	WER %
Baseline	78	557.3	0.47	1	11.1
BLAS	78	259.76	1	2.15	11.1
SVD* 72	72	254.9	1.02	2.19	11.1
SVD* 60	60	255.4	1.02	2.18	11.1
SVD* 48	48	258.22	1.01	2.16	10.9
SVD* 38	38	275.99	0.94	2.02	10.9
SVD* 28	28	318.80	0.81	1.75	11.9

Observations:

1. BLAS is 2.15 times faster than the Baseline.
2. Speedup due to the low-rank approximation method is in-significant.
3. As the rank decreases, computational performance decreases and recognition performance improves upto a point. Beyond one particular rank, both recognition and computational performance degrades rapidly.

5.2.2 Performance with Multi-Threaded BLAS

In Table 5, we report results with multi-threaded implementation of matrix-matrix multiplication using GotoBLAS2 v1.13 library.

Observations:

1. Computational performance degrades as rank decreases for single threaded as well as multi-threaded implementations.
2. Marginal speedups even after using multiple threads (for the same rank). E.g., For BLAS scheme, speedup of 1.13 for 8-threaded BLAS compared to single threaded BLAS.

5.2.3 Performance with Streaming Input

In Table 6, we report results with multi-threaded implementation of matrix-vector multiplication *sgemv* using GotoBLAS2 v1.13 library.

Observations:

1. Unlike results in Table 2, performance improves significantly as rank goes lower for low-rank matrix approximations for single thread implementations. However, after a certain point rank-reductions lead to degrade performance.
2. We get speedup in all cases compared to Baseline. Using single threaded BLAS with *sgemv* is 1.37 times faster than Baseline.
3. Speedups upto 1.61 for BLAS and upto 1.22 for SVD* 38 using 8-threads. Compared to results in Table 5, this shows that *sgemv* scales better than *sgemm* for our application.
4. No significant improvement in performance using multiple threads due to low-rank factorizations.

5.3 Performance Analysis

Table 7 shows three major phases of computation in Sphinx and their execution times for RM1 task. ALC_Orig is the original acoustic likelihood computation time (mgau_eval routine). We can see total ALC as consisting of three parts. (1) Evaluating Gaussian likelihoods $[\mathbf{X^2}|\mathbf{X}] \cdot [\frac{\mathbf{S}}{\mathbf{M}}]$ (2) Adding the constant component **C** and (3) Logarithmic addition to calculate total mixture score. For **Baseline** all ALC is ALC_Orig phase. For BLAS scheme we evaluate part (1) of ALC_Orig separately and fully and is called ALC_Precalc, because for each utterance all the likelihoods for all Gaussians for all feature vectors of that utterance are calculated at

Table 5 Sphinx 3.7 with Multithreaded BLAS on TIMIT task.

Scheme	1 thread	2 threads	4 threads	8 threads
Baseline	557.3			
BLAS	259.76	245.61	235.00	229.55
SVD *72	254.9	246.14	235.51	231.65
SVD *60	255.4	248.48	239.4	237.5
SVD *48	258.22	258.07	248.14	247.2
SVD *38	275.99	277.2	271.7	268.72
SVD *28	318.80	324.83	320.8	319.9

Table 6 Sphinx 3.7 with Multithreaded BLAS on TIMIT task on streaming input.

Scheme	1 thread	2 threads	4 threads	8 threads
Baseline	557.3			
BLAS	407.42	309.87	277.51	252.64
SVD *72	385.82	309.01	275.77	258.15
SVD *60	372.17	302.56	275.83	261.24
SVD *48	341.32	304.96	277.7	264.45
SVD *38	348.58	320.34	300.35	286.07
SVD *28	377.29	360.69	345.45	335.98

Author's personal copy

Table 7 Sphinx 3.7 phase profiling on RM1 task.

Scheme	ALC_Precalc (a)	ALC_Orig (b)	Search (c)	Total (a+b+c)
Baseline	0	100.9	22.1	123
BLAS	38.25	22.6	24.85	85.7
FNMF-(9,4)	30.8	23.03	24.47	78.3
FSVD-(18,34)	30.1	24.92	29.63	84.65
SVD* 38	25.5	24.34	25.66	75.5

Table 9 Sphinx 3.7 phase profiling on TIMIT task.

Scheme	ALC_Precalc (a)	ALC_Orig (b)	Search (c)	Total (a+b+c)
Baseline	0	403.91	149.49	553.4
BLAS	49.5	81.39	125.37	256.26
SVD* 60	41.3	82.71	130.6	254.47
SVD* 48	35.32	84.6	138.9	258.82
SVD* 38	31.58	88.24	158.09	277.91

the start of decoding. The ALC_Orig phase for BLAS which computes the second and the third parts (parts (2) and (3) above) of the total ALC takes 22.6 seconds. Part (i) of ALC (ALC_Precalc) takes only 38.25 seconds, which was originally (ALC_Orig for Baseline − ALC_Orig for BLAS =) 78.3 seconds. Hence, evaluating 3,580 Gaussians per frame in the baseline approach is almost two times (78.3/38.25) slower than evaluating all 15,480 Gaussians per frame using the BLAS matrix multiplication.

Going from BLAS to SVD* 38 should reduce ALC_Precalc by a factor of 2 and going from BLAS to FSVD-(18,34) should reduce ALC_Precalc by factor of 78/52=1.5 according to the prediction in Section 2; but it gets reduced by a factor of 1.5 and 1.27 respectively in the experiments. This is due to the number of memory operations remaining nearly same while arithmetic operations get reduced by a factor of 2 and 1.5 respectively. This can be seen easily with a numerical example given below.

Consider multiplication of matrices of size (340 × 78) and (78 × 15,480). This size is typical of our application where $m=340$ is number of feature vectors and $2d=78$ is two times the feature vector dimension and $n=15{,}480$ is number of Gaussians. The number of arithmetic operations are $2\times340\times78\times15{,}480$. When we reduce the rank of Gaussian parameter matrix to half in the resulting $2\times340\times39\times15{,}480$ computation the number of arithmetic operations get reduced to half, but the number of memory operations does not get reduced by the same amount. This is because a large part of memory operations is for storing the result matrix, which is very large compared to operand matrices and does not reduce through rank reduction. This is the reason we do not get overall speedup of 2; instead, suppose we were reducing the number of Gaussians to half so as the arithmetic operations still remain half of the original, but this time the big resultant matrix also becomes half the size, so that memory operations are also reduced almost by half and we can expect a speedup of nearly 2. We have conducted an experiment which is essentially similar to performing the likelihood computation matrix multiplication in isolation and summarized the results in Table 8. Each matrix multiplication was repeated 600 times (number of test utterances) and the size of feature vector matrix was chosen uniformly randomly with mean 340 (average size of an utterance in the test set). As we can see from Table 8 reducing the rank to 38 from 78 leads to a speedup of 1.55 while reducing the number of Gaussians from 15,480 to 7,740 leads to a speedup of 2.02. Similarly, reducing the rank from 78 to 52 leads to a speedup of 1.3 while reducing the number of Gaussians from 15,480 to 10,320 leads to a speedup of 1.504. This explains the above mentioned effect of number of memory operations.

Table 9 shows the Sphinx phase profiling on TIMIT task. We can clearly see that as the rank decreases ALC_Precalc time decreases because of smaller matrix multiplication. However, the Search time increases significantly to compensate for the errors in the likelihoods. This increase in Search time is more pronounced compared to similar increase for RM1 task as shown in Table 7. This is due to the fact that out of 9,104 total pre-computed Gaussians, Sphinx reads 5,688 Gaussians/frame on an average and with 1,599 active HMMs/frame on an average. For RM1 task, on an average, it was 3,580 Gaussians/frame and 615 HMMs/frame.

Table 8 Matrix multiplication benchmark.

Scheme	Time	Speedup
(340 × 78) × (78 × 15,480)	34.68	1
(340 × 52) × (52 × 15,480)	26.58	1.3
(340 × 38) × (38 × 15,480)	21.95	1.55
(340 × 78) × (78 × 10,320)	23.05	1.504
(340 × 78) × (78 × 7,740)	16.82	2.02

6 Conclusion

Expression of acoustic likelihood computation problem as matrix-matrix multiplication with its efficient implementation in sequential as well as multi-threaded BLAS enables faster large vocabulary continuous speech recognition on current multi-core CPUs. We reduce the likelihood computation further by (1) direct low-rank approximation of the Gaussian

Author's personal copy

parameter matrix and (2) derivation of better factors of the Gaussian parameter matrix using training data by an optimum low-rank approximation-factorization of the likelihood matrix. These methods result in over 60 % speedup in computation on 1,138 word vocabulary RM1 task, with the latter resulting in significantly lesser increase in word error rate. On a larger 6,224 word vocabulary TIMIT task with smaller acoustic model, the latter method results in decrease in Baseline WER for appropriate ranks. Overall, our low-rank matrix approximation methods allow us to efficiently trade-off computational performance with recognition performance in LVCSR. Our methods can also be applied efficiently for multi-core CPUs and with real-time speech input.

Appendix A: Efficient Euclidean Distance Matrix Computation in Dynamic Time Warping (DTW) Using Matrix Multiplication

Computing pairwise distances in Euclidean space between two sets of vectors is a general problem in pattern classification and is often a key computation affecting the performance of speech recognition systems. The matrix multiplication based formulation presented in Section 2 for computation of Gaussian likelihoods in LVCSR can be applied to compute pairwise Euclidean Distances between the Test and Template utterances in Dynamic Time Warping (DTW) algorithm. We show that it results in saving approximately $\frac{1}{3}$ of computational operations without using any low-rank matrix approximations.

A.1 Dynamic Time Warping (DTW) Algorithm

Dynamic Time Warping (DTW) is a technique for optimally aligning variable length sequences of symbols based on some distance measure between the symbols [27]. In speech processing applications, each symbol is called a feature vector and we have two speech utterances (also called *template* and *test* utterances) which are variable length sequences of feature vectors. Typically, a feature vector is obtained by feature extraction algorithms [27] for 10ms of speech. Euclidean distance measure is widely applied for the feature vectors thus obtained. DTW in recursive form is given in Algorithm 1.

DTW can be implemented without recursion for efficient implementation. Non-recursive DTW can be seen as two distinct phases. (1) Computing the Euclidean distance matrix (2) Computing the accumulated distance matrix and extracting the least cost path. In the next subsection we show how we can reduce the computational operations in the first phase, i.e., Euclidean distance matrix computation.

Algorithm 1 float DTW(i,j)

{**X** is the template utterance, $\mathbf{x_i} \in \mathbf{X}, \mathbf{1} \leq \mathbf{i} \leq \mathbf{m}$}
{**Y** is a test utterance, $\mathbf{y_j} \in \mathbf{X}, \mathbf{1} \leq \mathbf{j} \leq \mathbf{n}$}
{D(i,j) is a function returning squared Euclidean distance between $\mathbf{x_i}$ and $\mathbf{y_j}$}
{acc_d[i,j] is an array for storing the cost of the optimal path from (1,1) to (i,j)}
{N(i,j) is a set of neighbors of (i,j). E.g., (i-1,j),(i-1,j-1) and (i,j-1) namely the *left*, *diag* and *down* neighbors}

if i == 1 and j==1 **then**
  acc_d [1,1] ← D(1,1)
  return D(1,1)
else
  acc_d [i,j] ← D(i,j) + min { DTW(i',j') | (i',j') ∈ N(i,j) }
  return acc_d [i,j]
end if

A.2 Efficient Euclidean Matrix Computation Using Matrix Multiplication

The distance computation problem can be stated as follows: Given a set **X** of m feature vectors (of dimension d) denoted by $\mathbf{x_i}$, $1 \leq i \leq m$ from *template* utterance and a set **Y** of n feature vectors denoted by $\mathbf{y_j}$, $1 \leq j \leq n$ from *test* utterance, we compute Euclidean distance for all pairs $(\mathbf{x_i}, \mathbf{y_j})$ feature vectors. For a pair, computing the distance results in a dot product operation among augmented vectors obtained from feature vectors. So, the entire distance computation problem is a matrix-matrix multiplication problem among matrices obtained from feature vectors.

The squared Euclidean distance between two vectors **x** and **y** can be written is:

$$\mathbf{d}(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^\top (\mathbf{x} - \mathbf{y}) = \sum_{i=1}^{d} (x_i - y_i)^2 \tag{9}$$

The total number of arithmetic operations (addition and multiplication) for computing a dot product is $3d$. So, the total number of operations for computing distances for all pairs $(\mathbf{x_i}, \mathbf{y_j})$ forming the distance matrix **D** is

$$3mnd \tag{10}$$

If we expand the quadratic term $(x_i - y_i)^2$, we can see that the Euclidean distance is a sum of three

Author's personal copy

different terms: (1) $\sum_{i=1}^{d} x_i^2$ (2) $\sum_{i=1}^{d} y_i^2$ and (3) -2 $\times$ dot product of vector $\mathbf{x}$ with vector $\mathbf{y}$. Now, let us consider the sequence of feature vectors in template utterance , i.e., $\mathbf{x_1}$ to $\mathbf{x_m} \equiv \mathbf{X}$, a $m \times d$ matrix where each row is a feature vector, and similarly $\mathbf{Y}$ is a $d \times n$ matrix of columns of feature vectors from the test utterance, then the matrix of Euclidean distances can be expressed as:

$$\mathbf{D} = -\mathbf{2XY} + \mathbf{X^2} + \mathbf{Y^2} \tag{11}$$

$\mathbf{X^2}$ is a rank-1 matrix of size $m \times n$ where $\mathbf{X^2_{ij}} = \mathbf{x_i}^\top \mathbf{x_i}$. $\mathbf{Y^2}$ is a rank-1 matrix of size $m \times n$ where $\mathbf{Y^2_{ij}} = \mathbf{y_j}^\top \mathbf{y_j}$.

The number of arithmetic operations for computing $\mathbf{XY}$ is $2mnd$ (i.e., matrix multiplication). Multiplying $\mathbf{XY}$ with scalar -2 leads to extra mn multiplications. $\mathbf{X^2}$ requires computing $\mathbf{x_i}^\top \mathbf{x_i}$ where $1 \leq i \leq m$, requiring $2d$ operations. There are m vectors in $\mathbf{X}$ so total $2md$ operations. Similarly computing $\mathbf{Y^2}$ will take $2nd$ operations. So the total number of arithmetic operations for computing D is

$$2mnd + mn + 2md + 2nd \tag{12}$$

compared to the original $3mnd$ operations. Thus the condition for saving arithmetic operations is

$$3mnd > 2mnd + mn + 2d(m+n) \tag{13}$$

that is

$$d > \frac{1}{1 - 2\left(\frac{1}{m} + \frac{1}{n}\right)} \tag{14}$$

which makes sense if

$$\left(\frac{1}{m} + \frac{1}{n}\right) < \frac{1}{2} \tag{15}$$

which is true for all but very small values of m and n. Putting $m = n$ suggests $m > 4$.

The number of operations saved are

$$mnd - mn - 2d(m+n) \tag{16}$$

$3mnd$ being the original number of computations, the fraction f of original operations saved is

$$f = \frac{1}{3} - \frac{1}{3d} - \frac{2}{3}\left(\frac{1}{m} + \frac{1}{n}\right) \tag{17}$$

f approaches to $\frac{1}{3}$ as m, n and d increases. In our context, typical values of m and n are few hundreds to few thousands and d is typically 39. Thus the fraction of computations saved is $\sim \frac{1}{3}$, that is $\sim mnd$ operations.

Note that in Eq. 12, mn multiplications are required for computing $-\mathbf{2}(\mathbf{XY})$, if we compute $(-\mathbf{2X})\mathbf{Y}$ instead we will incur md multiplications, similarly for computing $\mathbf{X}(-\mathbf{2Y})$, we incur nd operations. Performing the same analysis by putting md in place of mn in Eq. 12, we can derive

$$f = \frac{1}{3} - \frac{1}{n} - \frac{2}{3m} \tag{18}$$

Thus the fraction of total operations saved only depends on m and n. The price to be paid is the induced error in the precision of $-\mathbf{2XY}$ matrix due to different order of evaluation.

A.3 Performance Results

In Table 10, we measure the impact of matrix-multiplication based Euclidean distance matrix computation for DTW on Intel Xeon E5440 CPU @ 2.83 GHz. For efficient matrix-multiplication we have used GotoBLAS2 v1.13 library. We use only single core for the computation. **Baseline:Total** indicates total running time of the non-recursive baseline DTW algorithm complied with -O2 optimization of gcc 4.2. **Matrix:Total** indicates the total running time of non-recursive DTW algorithm, where Euclidean distances are computed using matrix-multiplication approach. For our experiments, we generate the test and template utterances randomly. Each utterance is a sequence of 39 dimensional vectors. **Size** shows the length (in no. of 39 dimensional vectors) of the template and test utterances. Without loss of generality we have taken identical lengths for both test and template utterance. We can see that matrix multiplication based approach for distance computation brings substantial speedups for overall DTW computation. As we increase the size of utterances speedup decreases, due to more time spent on dynamic programming based distance accumulation phase.

In Table 11, we measure time for only Euclidean distance computation phase. **Baseline:DistCalc** shows time spent in distance computation in Baseline version and **Matrix:DistCalc** shows time spent in distance computation using matrix-multiplication. We get the

Table 10 Total execution time in seconds for DTW with traditional Euclidean distance matrix computation versus matrix-multiplication based computation.

Size	Baseline:Total	Matrix:Total	Speedup
5,000	1.639	0.504	3.25
10,000	6.917	2.365	2.92
15,000	16.763	6.493	2.58
20,000	37.007	18.951	1.95

Author's personal copy

Table 11 Performance comparison of traditional Euclidean distance matrix computation versus matrix-multiplication based Euclidean distance matrix computation, time in seconds.

Size	Baseline:DistCalc	Matrix:DistCalc	Speedup
5,000	1.382	0.247	5.60
10,000	5.538	0.986	5.62
15,000	12.493	2.223	5.62
20,000	22.047	3.991	5.52

speedup of 5.6 even as we scale the problem size. This confirms our theoretical analysis that, irrespective of the size of template and test utterances we are going to save a constant fraction $\frac{1}{3}$ of arithmetic operations. However, in practise speedup achieved is much more due to the fact that BLAS implementation improves the cache memory utilization too.

Appendix B: Comparison with Principal Component Analysis (PCA)

In this section, we compare and discuss our method of deriving low-rank factors of the Gaussian parameters by optimum low-rank approximation of $\mathbf{G}$ (henceforth the svd-augmented method) with other feature space rank reduction method PCA as described in [23].

B.1 Computational Performance Comparison

We will assume that both the methods, PCA and svd-augmented are implemented in matrix-matrix multiplication manner, described in this paper. Let $\mathbf{X}$ be a feature vector matrix of size $m \times d$ where rows containing individual feature vectors and let there be n Gaussians. The baseline number of arithmetic operations using matrix multiplication approach is $4mnd$ because we require two matrix multiplications of size $(m \times d) \times (d \times n)$ to compute the likelihood matrix or equivalently one $(m \times 2d) \times (2d \times n)$ multiplication.

When the rank gets reduced to $d - k$, the computation will accordingly reduce to $4mn(d - k)$.

We will analyze algorithms in terms of *overhead* and *savings*. The ideal *savings* will correspond to the fraction of the rank reduced, i.e., $4mnk$. But to reduce the features to the lowered rank, we will have to apply some transformation and thus, it is the *overhead*. It is also assumed that $n >> m$, otherwise the *overhead* may become greater than *savings*. The analysis of at what point the *overhead* will dominate the *savings* is performed in Section 2.2.

The computational overhead of PCA requires transforming $\mathbf{X}$ by a factor of size $d \times d - k$ where $0 < k \leq d$. The remaining computation then becomes a $(m \times 2d - 2k) \times (2d - 2k \times n)$, which is $4mn(d - k)$ operations.

Since svd-augmented method operates on augmented feature vectors $[\mathbf{X^2}|\mathbf{X}]$ of size $m \times 2d$, we have to multiply it with a factor of size $2d \times 2k$, to get it in the form $m \times 2d - 2k$. From, then on it is the same $(m \times 2d - 2k) \times (2d - 2k \times n)$ multiplication.

The overhead in case of PCA is: $(m \times d) \times (d \times d - k)$, that is $2md(d - k)$ operations. In case of svd-augmented method, the overhead is: $(m \times 2d) \times (2d \times 2d - 2k)$, that is $8md(d - k)$ operations.

So, svd-augmented method incurs four times the overhead than PCA. However, since the computation savings is $4mnk$, this overhead would be a small fraction of it, given $n >> m, d$.

There is one minor point still left. In case of svd-augmented method, before going for matrix-multiplication, we have to compute element wise square of $\mathbf{X}$ that is $\mathbf{X^2}$, which takes m^2d^2 operations. In case of PCA, we first reduce the rank of $\mathbf{X}$, so we have to compute squares on reduced size matrix of $m \times (d - k)$, that is $m^2(d - k)^2$ operations.

So, The original baseline computation:

$$m^2d^2 + 4mnd$$

Computation after rank reduction of k for PCA:

$$m^2(d - k)^2 + 2md(d - k) + 4mnk$$

Computation after rank reduction of $2k$ for svd-augmented:

$$m^2d^2 + 8md(d - k) + 4mnk$$

B.1.1 Discussion

One point here is that we have implicitly assumed the equivalence of reduction of rank by k in feature space with that of reduction of rank $2k$ in augmented feature space of dimension $2d$. Strictly speaking, we can not compare PCA with the svd-augmented because it operates on the augmented feature space. For ideal comparison with svd-augmented method, PCA have to be carried out on the augmented features, but then we can not use existing training algorithms, because that would mean we are estimating means for vectors of type $[\mathbf{X^2}|\mathbf{X}]$, that will change the model and the recognizer completely. So, to work on augmented feature space PCA, we need new training algorithms that train the co-efficients that finds some equivalence in multivariate Gaussian modeling of feature space. We have not explored this direction.

Author's personal copy

B.2 Recognition Performance Comparison with a Version of PCA

The svd-augmented method is an unsupervised method similar to PCA. PCA retains the directions in which the variance in the training data is maximum. Let us denote the matrix which transforms the feature space into lower dimensional one as described above as **P** which is $d \times d - k$. For the transformed features, **XP** the means will become **MP** and the covariances will be $\mathbf{P\Sigma P^T}$. Here the covariance matrices will become full covariance. So, we need to estimate the diagonal covariances once again.

If we project the reduced rank features back on to d dimensional space, the resulting features $\mathbf{XPP^T}$ will give minimum sum of square errors (Frobenius norm) with **X**. Now we can compare this version of PCA with svd-augmented method which minimizes sum of square errors in the likelihood matrix $\left[\mathbf{X^2}|\mathbf{X}\right] \cdot \left[\frac{\mathbf{S}}{\mathbf{M}}\right]$ for a given Gaussian parameter model $\left[\frac{\mathbf{S}}{\mathbf{M}}\right]$. Table 12 shows the WER against effective rank comparison. We can observe that as the rank reduces, svd-augmented performs better as PCA method's WER increases sharply. We think that, this happens because as we lower the rank, more and more of useful information carrying dimensions are discarded in case of the PCA method. While, since the svd-augmented method approximates likelihoods, it contains the information of the higher dimensions and that is encoded in its augmented feature transform matrix of size $2d \times 2d - 2k$. Intuitively, we can say that the svd-augmented method captures more information at the cost of increased *overhead*.

B.3 Summary

- PCA method requires storing different factors for different ranks separately while we can select the top-k rows(columns) for selecting optimum factors for rank-k. This is an elegant way of efficiently implementing an LVCSR system where we can trade off computation with accuracy dynamically.
- The svd-augmented method approximates the baseline model, while PCA retrains the models for lower rank.
- The svd-augmented method operates on augmented feature space while PCA transforms original feature space.
- The svd-augmented method works better than the PCA as the effective rank reduces because it captures the highest variances in the likelihoods, and errors in likelihoods have direct impact on word error rate in LVCSR systems. However, the svd-augmented method incurs three times more computational overhead compared to PCA, which is still small because the original overhead of PCA is small, given the number of Gaussians in the model are large.

Table 12 Comparison of percent word error rates on RM1 task on Sphinx 3.7.

Rank	PCA-version	svd-augmented
30	3.3	3.5
24	3.4	3.5
19	4.3	4.0
15	6.8	5.0
10	32.5	9.8

The baseline word error rate is 3.2 %.

References

1. Agaram, K., Keckler, S.W., Burger, D. (2001). A characterization of speech recognition on modern computer systems. In: *IEEE international workshop on workload characterization, 2001* (pp. 45–53).
2. Krishna, R., Mahlke, S., Austin, T. (2002). Insights into the memory demands of speech recognition algorithms. In: *Proceedings of the 2nd annual workshop on memory performance issues*.
3. Lai, C., Lu, S.-L., Zhao, Q. (2002). Performance analysis of speech recognition software. In: *Proceedings of the 5th workshop on computer architecture evaluation using commercial workloads*.
4. Povey, D., Kingsbury, B., Mangu, L., Saon, G., Soltau, H., Zweig, G. (2005). Fmpe: Discriminatively trained features for speech recognition. In: *IEEE international conference on acoustics, speech, and signal processing, 2005* (Vol. 1, pp. 961–964).
5. Chan, A., Ravishankar, M., Rudnicky, A., Sherwani, J. (2004). Four-layer categorization scheme of fast gmm computation techniques in large vocabulary continuous speech recognition systems. In: *Proceedings of international conference on spoken language processing, Interspeech 2004*.
6. Bocchieri, E. (1993). Vector quantization for the efficient computation of continuous density likelihoods. In: *IEEE international conference on acoustics, speech, and signal processing, 1993* (Vol. 2, pp. 692–695).
7. Lee, A., Kawahara, T., Shikano, K. (2001). Gaussian mixture selection using context-independent hmm. In: *IEEE international conference on acoustics, speech, and signal processing, 2001* (Vol. 1, pp. 69–72).
8. Ravishankar, M., Bisiani, R., Thayer, E. (1997). Sub-vector clustering to improve memory and speed performance of acoustic likelihood computation. In: *Proceedings of European conference on speech communication and technology, Eurospeech 1997*.

Author's personal copy

9. Pellom, B.L., Sarikaya, R., Hansen, J.H.L. (2001). Fast likelihood computation techniques in nearest-neighbor based search for continuous speech recognition. *IEEE Signal Processing Letters, 8*(8), 221–224.
10. Saraclar, M., Riley, M., Bocchieri, E., Goffin, V. (2002). Towards automatic closed captioning: low latency real time broadcast news transcription. In: *Proceedings of international conference on spoken language processing, interspeech 2002*.
11. Cardinal, P., Dumouchel, P., Boulianne, G., Comeau, M. (2008). Gpu accelerated acoustics likelihood computations. In: *Proceedings of international conference on spoken language processing, interspeech 2008*.
12. Dixon, P.R., Caseiro, D.A., Oonishi, T., Furui, S. (2007). The titech large vocabulary wfst speech recognition system. In: *IEEE workshop on automatic speech recognition & understanding, 2007* (pp. 443–448).
13. Dixon, P.R., Oonishi, T., Furui, S. (2009). Harnessing graphics processors for the fast computation of acoustic likelihoods in speech recognition. *Computer Speech & Language, 23*(4), 510–526.
14. Chong, J., Gonina, E., Yi, Y., Keutzer, K. (2009). A fully data parallel wfst-based large vocabulary continuous speech recognition on a graphics processing unit. In: *Proceedings of international conference on spoken language processing, interspeech 2009*.
15. Dixon, P.R., Oonishi, T., Furui, S. (2009). Fast acoustic computations using graphics processors. In: *IEEE international conference on acoustics, speech and signal processing, 2009* (pp. 4321–4324).
16. Cai, J., Bouselmi, G., Fohr, D., Laprie, Y. (2008). Dynamic gaussian selection technique for speeding up hmm-based continuous speech recognition. In: *IEEE international conference on acoustics, speech and signal processing, 2008* (pp. 4461–4464).
17. Morales, N., Gu, L., Gao, Y. (2008). Fast gaussian likelihood computation by maximum probability increase estimation for continuous speech recognition. In: *IEEE international conference on acoustics, speech and signal processing, 2008* (pp. 4453–4456).
18. Hennessy, J.L., & Patterson, D.A. (2003). *Computer architecture: A quantitative approach*. San Francisco: Morgan Kaufmann Publishers Inc.
19. You, K., Lee, Y., Sung, W. (2007). Mobile cpu based optimization of fast likelihood computation for continuous speech recognition. In: *IEEE international conference on acoustics, speech and signal processing, 2007* (Vol. 4, pp. IV–985–IV–988).
20. Choi, Y., You, K., Choi, J., Sung, W. (2010). A real-time fpga-based 20000-word speech recognizer with optimized dram access. *IEEE Transactions on Circuits and Systems I: Regular Papers, 57*(8), 2119–2131.
21. Dongarra, J.J., Croz, J.D., Hammarling, S., Duff, I.S. (1990). A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software, 16*(1), 1–17.
22. Duda, R.O., Hart, P.E., Stork, D.G. (2000). *Pattern classification*. Wiley-Interscience Publication.
23. Paliwal, K.K. (1992). Dimensionality reduction of the enhanced feature set for the hmm-based speech recognizer. *Digital Signal Processing, 2*, 157–173.
24. Eckart, C., & Young, G. (1936). The approximation of one matrix by another of lower rank. *Psychometrika, 1*(3), 211–218.
25. Lee, D.D., & Seung, H.S. (2000). Algorithms for non-negative matrix factorization. In: *Neural Information Processing Systems, NIPS 2000* (pp. 556–562).
26. Huang, X., Acero, A., Hon, H.-W. (2001). *Spoken language processing: A guide to theory, algorithm, and system development* (1st ed.). Upper Saddle River: Prentice Hall PTR.
27. Rabiner, L., & Juang, B.-H. (1993). *Fundamentals of speech recognition*. Englewood Cliffs: Prentice-Hall.

Mrugesh R. Gajjar received the BE degree in computer engineering from Dharmsinh Desai University, Nadiad in 2002, and the MTech degree in information and communication technology from Dhirubhai Ambani Institute of Information and Communication Technology, Gandhinagar, in 2004. He has total 8 years of teaching, research and industrial experience in various capacities. His research interests include high performance computing, signal processing systems and performance optimization. He is currently with parallel systems group at Siemens Corporate Research and Technologies, Bangalore.

T. V. Sreenivas is a professor at Indian Institute of Science, Bangalore, teaching and researching in the area of signal processing for speech and audio applications. He has contributed to over 150 research papers in international conferences and journals and 3 patents. He has introduced and taught several new post-graduate courses at IISc. His academic activity includes interactions with several research groups around the world and has been a visiting faculty member at Norwegian University

Author's personal copy

of Science and Technology (NTNU), Trondheim, Norway; Marquette University, Milwaukee, USA; Fraunhofer Institute for Integrated Circuits (IIS), Erlangen, Germany; Fraunhofer Institute for Digital Media Technologies (IDMT), Ilmenau, Germany; Technical University of Ilmenau, Germany; Royal Institute of Technology (KTH), Stockholm, Sweden; and Griffith University, Brisbane, Australia. He has also been a faculty entrepreneur, co-founding "Esqube Communication Solutions Pvt. Ltd.", a technology startup company in Bangalore. His current research interests are cognitive models for speech/audio processing, compact neural representation of speech/audio (compressive sensing), multi-microphone speech/audio processing, robotic applications of speech/audio (CASA). He is a senior member of IEEE, fellow of IETE, founding member and past chairman of IEEE Signal Processing Society Bangalore Chapter, co-founder of the series of "Winter Schools on Speech and Audio Processing" (WiSSAP) and contributor to IEEE-SPCOM series of conferences. He graduated from Bangalore University in 1973, obtained M.E. from Indian Institute of Science in 1975, and Ph.D. from Indian Institute of Technology, Bombay in 1981, working as research scholar at Tata Institute of Fundamental Research, Bombay.

R. Govindarajan received the BSc degree in mathematics from Madras University in 1981, and the BE degree in electronics and communication, and the PhD degree in computer science from the Indian Institute of Science, Bangalore in 1984 and 1989, respectively. He has held postdoctoral research positions at the University of Western Ontario, Canada, and McGill University, Canada, and faculty position in the Department of Computer Science, Memorial University of Newfoundland, Canada. He has held visiting faculty positions at the University of Delaware and Arizona State University. Since 1995, he has been with the Supercomputer Education and Research Centre and the Department of Computer Science and Automation, Indian Institute of Science, where he is currently a professor and the chairman of the Supercomputer Education and Research Centre. He is a senior member of the IEEE and the IEEE Computer Society.