RAPID LIKELIHOOD CALCULATION OF SUBSPACE CLUSTERED GAUSSIAN COMPONENTS

A. Aiyer, M.J.F. Gales and M.A. Picheny

IBM TJ Watson Research Center
Yorktown Heights, NY
anuaiyer@ee.stanford.edu, mjfg@eng.cam.ac.uk, picheny@us.ibm.com

ABSTRACT

In speech recognition systems, computing the likelihoods of the acoustic models is an intensive task. One approach to reduce this cost is to use subspace distributed clustering HMM. Here individual Gaussian components are stored as indices to, and their likelihoods computed from, a set of subspace Gaussian components. This paper examines a scheme for reducing the computational cost of the likelihood calculation when such an HMM system is used. The proposed method identifies and stores frequently occurring partial sums called meta-atom elements and thus avoids computing them repeatedly. The resultant savings in the number of additions is 50% when all Gaussian components are computed or 20% when a Gaussian selection scheme is used.

1. INTRODUCTION

One of the most intensive and repeated tasks in a speech recognition system is computing the likelihoods of the acoustic models. For any speech recognition system to be practically useful, these computations need to be done very quickly, while keeping the memory required to store the models as small as possible. One commonly used approach which reduces both the computational cost and results in a compact model is the use of tying, for example state-tying [6] and semi-continuous HMMs [3]. This paper examines a scheme to improve the speed of the likelihood calculation in a particular type of tied system called subspace distributed clustering HMM (SDCHMM) [5].

SDCHMMs may be trained in one of two ways. In the first method, a standard continuous density HMM system is trained. The space is then split into K subspaces (or streams). For each subspace the Gaussian components from all the states are clustered together to some set of canonical Gaussian components in that subspace [5] called *subspace Gaussian components* or *atoms*. In the second method, given the set of subspaces and the tying structure, the subspace Gaussian components may be trained in a EM-fashion [4].

Regardless of which training method is used, the feature vectors in SDCHMMs are split into a set of subspaces. Each subspace is modeled by a set of atoms. Each model Gaussian component consists of weights and, rather than a mean vector and variance vector, an index to each of the subspaces indicating the particular atom of that subspace. The complete model set consists of a set of atoms and indices.

The work in this paper does not consider the generation or training of SDCHMMs, but rather given a trained system how to rapidly compute the likelihoods for each Gaussian component in the system.

This paper is organized as follows. The next section describes the basic form of the SDCHMM and how groupings within the Gaussian components may be used to speed up the computation. Then a brief description of two algorithms for determining appropriate subspace groupings is given. Finally the percentage speed-ups on a large vocabulary speaker-independent speech recognition system are described.

2. SDCHMM LIKELIHOOD CALCULATION

The calculation of likelihoods in SDCHMMs is far more efficient than that in standard HMMs. Each Gaussian component is represented as a vector of K indices that indicate the particular set of subspace Gaussian components, or atoms, that are to be used to calculate the likelihood and the diagonal covariance matrix.[1] Thus

$$\mathcal{L}(\mathbf{o};\mathbf{v}^{(m)}) = \prod_k \mathcal{N}(\mathbf{o}^{(k)};\mu^{(kv_k^{(m)})},\Sigma^{(kv_k^{(m)})}) \tag{1}$$

where $\mathbf{v}^{(m)}$ is the vector of indices associated with Gaussian component m, $\mathbf{o}^{(k)}$ is the observation vector in the k^{th} subspace, and $\mu^{(kv_k^{(m)})}$ and $\Sigma^{(kv_k^{(m)})}$ are the mean and variance of the atom indicated by $v_k^{(m)}$. For each Gaussian component in the system it is only necessary to store the index vector $\mathbf{v}^{(m)}$. The standard technique for calculating the likelihood consists of two sections.

1. **Subspace component calculation**: For each of the K subspaces and N atoms per subspaces calculate the subspace log-likelihood. This fills a $K \times N$ table of log-likelihoods, the *atom table*.

Anuradha Aiyer performed this work while a summer student at IBM Almaden Research Center. Her current address is Information Systems Laboratory, Stanford University.

Mark Gales is now at Cambridge University.

[1] There is an inherent assumption that either diagonal covariance matrices are being used, or a block diagonal covariance structure where blocks do not cross subspace boundaries are being used. In this work diagonal covariance matrices are used throughout.

2. **Gaussian component calculation**: Using the atom table the likelihood of each of the M Gaussian components may be calculated by summing the K atoms indicated by the index vector $\mathbf{v}^{(m)}$.

Thus

$$\log\left(\mathcal{L}(\mathbf{o};\mathbf{v}^{(m)})\right) = \sum_k a_{kv_k^{(m)}} \tag{2}$$

where a_{kn} is the (k,n) element of the atom table,

$$a_{kn} = \log(\mathcal{N}(\mathbf{o}^{(k)};\mu^{(kn)},\Sigma^{(kn)})) \tag{3}$$

A simple scheme for computing the Gaussian components calculates each component independently, requiring $K-1$ additions per component. Such a scheme assumes that there are no regular patterns in the index vectors. This paper attempts to find the regular patterns and use them to reduce the number of additions.

The process of using the regular patterns to reduce the computational load is best illustrated by a simple example. Consider a 6 Gaussian components with 4 subspaces and 4 atoms per subspace (i.e. $M=6, K=4, N=4$).

$$\begin{array}{lclcccccc}
\mathbf{v}^{(1)} & = & < & 2 & 4 & 1 & 4 & > \\
\mathbf{v}^{(2)} & = & < & 4 & 1 & 2 & 2 & > \\
\mathbf{v}^{(3)} & = & < & 3 & 2 & 4 & 3 & > \\
\mathbf{v}^{(4)} & = & < & 1 & 4 & 3 & 1 & > \\
\mathbf{v}^{(5)} & = & < & 2 & 3 & 1 & 4 & > \\
\mathbf{v}^{(6)} & = & < & 2 & 4 & 1 & 4 & >
\end{array}$$

A naive computation of all the Gaussian components requires $M(K-1)$, 18, additions. However it is clear that there are distinct patterns in the Gaussian component indices. Consider components 1, 5 and 6, and notice that subspaces 1, 3 and 4 use the same atoms. If a *meta-atom element* consisting of the pre-computed sum of these atoms is used, it will be more computationally efficient. Thus we can write

$$\begin{array}{lclccc}
\mathbf{v}^{(1)} & = & < & c_1 & 4 & > \\
\mathbf{v}^{(5)} & = & < & c_1 & 3 & > \\
\mathbf{v}^{(6)} & = & < & c_1 & 4 & >
\end{array}$$

where

$$c_1 = a_{12} + a_{31} + a_{44}$$

This suggests that we should compute c_1 only once and use it repeatedly in $\mathbf{v}^{(1)}$, $\mathbf{v}^{(5)}$, and $\mathbf{v}^{(6)}$. Precomputing c_1 results in a one-time overhead of 2 additions. But now each of $\mathbf{v}^{(1)}$, $\mathbf{v}^{(5)}$, and $\mathbf{v}^{(6)}$ takes 1 addition as opposed to 3 additions. This results in an overall saving of $(3-1)\times 3 - 2 = 4$ additions. The savings in computing the Gaussian components dominates the overhead of computing the meta-atom elements.

Another interesting aspect of using meta-atom elements is the memory requirements of the system. Assuming that the number of subspace components is less than 256, then the index for each Gaussian component, $\mathbf{v}^{(m)}$, can be stored in K bytes. However by using meta-atom elements the number of indices stored for each component is reduced, but with the overhead of storing the meta-atom element. The meta-atom element simply costs b_i bytes, where b_i is the number of subspaces associated with element i, to store the indices to the atoms. The storage of the indices to the meta-atom elements is slightly more complicated since the number of meta-atom elements will typically exceed 256, it will require 2 bytes to store the index value. Thus provided b_i is greater than 2 and the number of Gaussian components using the index is large enough, it is theoretically possible to *save* memory by adding meta-atom elements.[2]

In the toy example shown it was simple to identify an appropriate meta-atom element to generate. For larger systems, for example the one considered in this paper where $M = 43444$, $K = 40$ and $N = 64$ it is not so simple to find the appropriate groupings for the meta-atom elements. Indeed it is not clear *a-priori* whether such groupings exist. As an initial investigation the maximum number of common indices for each Gaussian component pairings was examined[3]. using an exhaustive search. The histogram of resulting maximum common indices is shown in figure 1. There are a few interesting observations from the curve. The average number of common patterns is around twelve. This is far greater than the number of common patterns expected by chance, which is around 0.77. From this plot there do appear to be useful patterns in the Gaussian components that may be exploited to reduce the computational cost.

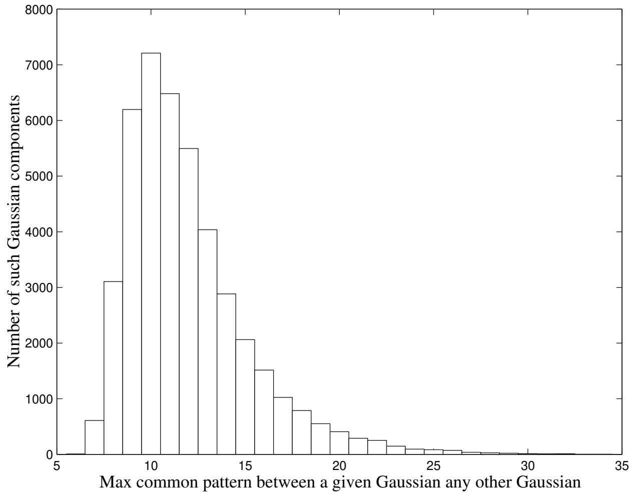

Figure 1: Histogram showing the length of the longest pattern between a given Gaussian component and any other Gaussian component.

[2]To practically realize these gains greatly complicates the storage of the indices, $\mathbf{v}^{(m)}$. It is necessary to have a mechanism to know which bands use meta-atom elements and which use standard elements. These problems are not addressed in this paper.

[3]A similar investigation was performed in [4] at a state level to examine relationships between subspace components and phonetic classes.

3. META-ATOM ELEMENT GENERATION

The problem is how to generate the optimal, in the sense of maximum computation reduction, set of meta-atom elements associated with a particular SDCHMM set. To obtain the strict optimal set is an NP-hard problem, which for the size of tasks being considered here is impractical. As a practical alternative, greedy algorithms that make locally optimal decisions.

Two approaches are considered for generating the set of meta-atom elements, one strictly greedy, the other probabilistically greedy.

3.1. Greedy Algorithm

Define the complete, combined set of meta-atom elements and the original elements of the atom table as the *set of elements*, $\mathbf{e}$. Each element will have a *weight* associated with it, that is the number of Gaussian components that have an index to that element.

1. Initialize the set of elements, $\mathbf{e}$, with the $K \times N$ atom table. Store the weight and subspace of each element.
2. Select the element, $\hat{e}$, with the highest weight.
3. Select the member of $\mathbf{e}$, $\tilde{e}$, that gives the greatest reduction in the number of additions when combined with $\hat{e}$. To determine $\tilde{e}$, note that one need not search through all elements. It suffices to consider only those elements whose subspaces do not overlap with the subspaces of $\hat{e}$.
4. Create a new meta-atom element combining $\hat{e}$ and $\tilde{e}$. Calculate the weight and subspace of this new element and add it to $\mathbf{e}$.
5. Update the weights of $\hat{e}$ and $\tilde{e}$ to reflect those Gaussian components using the new meta-atom element.
6. Goto step 2 until convergence criterion satisfied.

During each iteration of the algorithm, a new meta-atom element is generated in one of three ways: 1) combining two of the original elements of the atom table, 2) one original atom element and one meta-atom element or 3) two meta-atom elements. Methods 2 and 3 allow for patterns containing more than 2 atoms to be discovered.

This algorithm makes highly local selections. First it assumes that the element with the largest weight will yield the greatest savings. Furthermore the second element to combine with is chosen based on immediate gain, rather than possible future gains. It is trivial to derive scenarios where this is highly suboptimal.

3.2. Probabilistically Greedy Algorithm

Rather than simply selecting the element with the greatest weight, $\hat{e}$, at stage (2) of the greedy algorithm, a probabilistic selection scheme may be used. The probability of selecting a particular element is made proportional to some function of the attributes of that element. For example

$$p_i \propto f(w_i, b_i) \tag{4}$$

where w_i is the weight of element i and b_i is the number of subspaces associated with that element. For the initial probabilistic greedy algorithm investigated here, p_i is simply made proportional to the weight w_i. That is, $\hat{e}$ is chosen with a probability proportional to its frequency of occurrence in the various Gaussian components.

4. INTERACTION WITH GAUSSIAN SELECTION

An inherent assumption in the generation of the meta-atom elements in the previous section is that all Gaussian components are calculated at all time instances. In the vast majority of practical systems this is not the case. For Viterbi-based decoders at each time instance it is only necessary to calculate Gaussian components for those states on active paths. Furthermore Gaussian selection(GS) schemes may also be used [2]. This may reduce the savings obtained by the greedy algorithm because only a subset of all components are computed.

It is possible to improve the greedy algorithm by introducing heuristics into stage (3) of the meta-atom element generation scheme. Now rather than selecting the element that yields the greatest gain assuming all are generated, an approximation to the expected gain given the particular GS scheme is used. For example the simplest heuristic is to weight the gains from Gaussian components in $\hat{e}$'s GS subset differently from Gaussian components in other subsets. This is because if two Gaussian components are in the same GS subset, either both are computed or neither is computed. These schemes are not investigated in this paper.

5. RESULTS

The system used to investigate the possible speed ups is a large vocabulary speaker-independent speech recognition system. There are 43444 Gaussian components assigned to around 3,500 states. A 40-dimensional feature-space is used. 40 subspaces are used (one per feature vector element) with 64 atoms per subspace. On a speaker-independent task this system gives around 11% word error rate.[4]

First, the greedy algorithm was used to determine the meta-atom elements. The number of additions saved and the memory cost in storing the meta-atom elements are in figure 2. For the top graph, the addition savings, the meta-atom elements are ordered according to when they were added to the set of meta-atom elements. From the graph it can be seen that even using the sub-optimal greedy scheme described here can save large numbers of additions. For the bottom graph the meta-atom elements are ordered according to memory savings.This was in order to illustrate the potential memory saving aspects of using meta-atom elements. Though not well illustrated, due to scale, adding some meta-atoms did give potential memory savings. Similar shape graphs were obtained for the probabilistic greedy

[4]This paper does not discuss word error rates in any detail since the implemented scheme is guaranteed not to increase the word error rate since the Gaussian component log-likelihoods are not affected. The system used to test the algorithms is on a 20000 word, perplexity 160, internal LVCSR task. The performance is only given to illustrate that a relatively "good" system was being used.

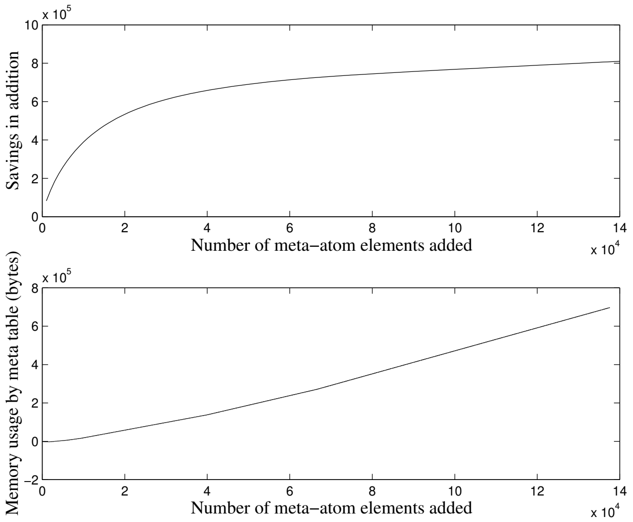

Figure 2: Results using the Greedy Algorithm. Top graph the sorted addition gains again meta-atom elements added. Bottom graph memory usage.

algorithm. Both schemes required additional memory to store the meta-atom elements. This indicates (and illustrated later) that the majority of meta-atom elements only have few subspaces associated with them.

Scheme	% Savings	Memory
Greedy	48.5%	732K
Prob. Greedy	48.7%	713K

Table 1: Summary of Results: Total number of additions saved and memory usage for each algorithm at the time of completion of algorithm

Table 1 shows a summary of the greedy and probabilistic greedy schemes in terms of the total percentage savings (compared to computing all Gaussian components in the SCDHMM system) and the total memory required. It may be seen that the two algorithms performed approximately the same and the probabilistically greedy scheme gives slightly greater gains and requires less memory. On examining the meta-atom elements, the probabilistic greedy search generated meta-atom elements using a greater number of subspaces, up to 16, in contrast to the greedy search where the largest number of subspaces for a particular meta-atom element was 8.

The results so far have been based on computing all the Gaussian components at every time instance. For practical schemes this is rarely done and GS schemes are used. Using GS the number of Gaussian components computed is reduced from 43444 to an average of around 1600 per time frame, less than 4% of the total. This would be expected to dramatically reduce the number of additions saved. On a speaker-independent task a saving of 20.1% in the number of additions is achieved using the greedy algorithm. Though significantly worse than the 50% savings achieved when computing all Gaussian components, it is greater than would be expected by reducing the number of computed Gaussian components by a factor of 25. The reason for this is that the GS scheme used is based on subsets of Gaussian components that are “close” in acoustic space. Since components that are close in acoustic space may be expected to share subspace components the loss in the number of additions is not as great as might be expected.

6. CONCLUSION

This paper has presented a simple scheme for increasing the speed of computing the likelihoods for SDCHMMs. SDCHMMs are useful since they yield a highly compact modelset and, compared to standard HMMs, have rapid likelihood calculations. This paper shows that these likelihood calculations may be made even faster by introducing *meta-atom* elements, or groups of subspaces. These meta-atom elements store frequently occurring partial sums and avoid computing them repeatedly. When all Gaussian components are computed, the number of additions is reduced by a factor of 2. When using a Gaussian selection scheme, a 20% reduction is achieved. The results presented are purely based on number of additions saved. Other issues, such as caching, are not considered, though these will, of course, impact the final speed improvements. Further work on using knowledge of the Gaussian selection scheme in the generation of the meta-atom elements will be performed.

REFERENCES

[1] L.R. Bahl, P.V. de Souza, P.S. Gopalkrishnan, D. Nahamoo, and M.A. Picheny. Performance of the IBM large-vocabulary continuous speech recognition system on the ARPA Wall Street Journal task. In *Proceedings ICASSP*, pages 41–44, 1995.

[2] E. Bocchieri. Vector quantization for efficient computation of continuous density likelihoods. In *Proc. ICASSP*, volume II, pages II–692–II–695, Minneapolis, 1993.

[3] X.D. Huang and M.A. Jack. Semi-continuous hidden Markov models for speech signals. *Computer Speech and Language*, 3:239–251, 1989.

[4] B. Mak. *Towards a Compact Speech Recogniser: Subspace Distribution Clustering Hidden Markov Model.* PhD thesis, Oregon Graduate Institute of Science and Technology, 1998.

[5] B. Mak and E. Bocchieri. Subspace distribution clustering for continuous observation density hidden Markov models. In *Proceedings Eurospeech*, pages 107–110, 1997.

[6] S.J. Young and P.C. Woodland. The use of state tying in continuous speech recognition. In *Proceedings Eurospeech*, pages 2207–2210, 1993.