Algorithms for Data-Driven ASR Parameter Quantization

Karim Filali, Xiao Li, Jeff Bilmes
{karim,lixiao,bilmes}@ssli.washington.edu

Depts of CS and EE, University of Washington
Seattle WA, 98195

UWEE Technical Report
Number UWEETR-2003-0010
June 2003

Algorithms for Data-Driven ASR Parameter Quantization

Karim Filali, Xiao Li, Jeff Bilmes
{karim,lixiao,bilmes}@ssli.washington.edu

Depts of CS and EE, University of Washington
Seattle WA, 98195

University of Washington, Dept. of EE, UWEETR-2003-0010
June 2003

Abstract

It is important to produce automatic speech recognition (ASR) systems that use as few computational and memory resources as possible, especially in low-memory/low-power environments such as for personal digital assistants. One way to achieve this is through parameter quantization. In this work, we compare a variety of novel subvector clustering procedures for ASR system parameter quantization. Specifically, we look at systematic data-driven subvector selection techniques based on entropy minimization, and compare performance primarily on an isolated word speech recognition task.

While the optimal entropy-minimizing quantization methods are intractable, several simple schemes including scalar quantization with separate codebooks per parameter and joint scalar quantization with normalization perform well in their attempt to approximate the optimal clustering.

1 Introduction

For certain applications, automatic speech recognition (ASR) will undoubtedly become the dominant human-computer interface methodology. For example, whenever hands are occupied (e.g., while driving), or where hand-based interfaces are bulky (using personal digital assistances (PDAs) or cell phones), ASR will undeniably succeed. Indeed, ASR is increasingly used on hand-held devices [8] — some PDA-based ASR systems are starting to appear commercially such as the IBM personal speech assistant [3] and the Microsoft MiPad [5] (others are listed in [8]). A number of wireless communication companies also launched their products integrated with ASR systems, like Motorala's voiceXML and Nokia 9000 series.

Compared to their wired brethren, these portable computing devices invariably have limited computational and memory resources and strict power consumption constraints. Therefore, as more functionality is pushed into and better performance is demanded of portable ASR systems, it becomes crucial to investigate power saving techniques. Several approaches can achieve this goal such as voltage modulation, computation reduction, optimization for special applications (small vocabulary recognition), modified decoding algorithms, and low-memory consumption. The last approach addresses the problem of limited storage (ASR systems use a significant amount of memory to store parameters, typically means and variances of multivariate Gaussian distributions or vector-quantized codebooks), the higher power consumption associated with higher memory usage and processor memory traffic [11], and computation [10].

A simple yet effective way to reduce the required resources with little effect on performance is to use fewer bits per parameter. This is done by further quantizing numerical representations well below the typical 32 or 64 bits per parameter used with the IEEE floating point standard. In the past, several techniques have been used to achieve such quantization. Scalar quantization [12] jointly clusters the individual elements of parameter vectors (means and diagonal covariances) in order to achieve lower memory requirements. Subvector clustering quantizes subsets of the parameter vectors. In most cases, however, the choice of the subvectors uses knowledge only of the type of features used; for example clustering MFCC's as one subvector, the first derivatives as a second subvector, or grouping each MFCC with its 1st and 2nd derivatives. In [2], 2-dimensional subvectors are formed using a greedy algorithm that chooses pairs that are most strongly correlated.

In this paper, we evaluate and compare a variety of novel methods for subvector quantization of parameters of continuous density hidden Markov model (HMM) ASR systems. Specifically, we look at systematic data-driven vector clustering techniques based on entropy minimization (equivalently mutual information maximization), and compare their performance on a isolated word speech recognition task [9]. While the optimal entropy-minimizing quantization method is intractable, we show that although several of our heuristic techniques are elaborate in their attempt to approximate the optimal clustering, simple scalar quantization using separate codebooks per parameter and joint scalar quantization with normalization perform surprisingly well.

In section 2, we describe our clustering algorithms and the subvector quantization techniques. Section 3 describes the speech corpus used and the experimental setup. In section 4 we show the memory-performance trade-off results of our experiments. Section 5 discusses the results and conclusions.

2 CLUSTERING ALGORITHMS

In the general problem of subvector quantization, we are given N vectors $v^{(i)}, i = 1, \ldots, N$ each of dimension D, which are to be quantized in some way. In this work, the N $v^{(i)}$s consist of the N means or N diagonal covariance matrices in a Gaussian-mixture HMM-based ASR system.[1] In subvector quantization, one decides upon M subsets $\{C_j\}_{j=1}^{M}$ of the index set $\mathcal{S} \triangleq \{1, 2, \ldots, D\}$, where $C_j \subseteq \mathcal{S}$ and where $C_j \cap C_m = \emptyset$ for all $j \neq m$ and $\bigcup_j C_j = \mathcal{S}$. For each of these subvectors, there are K_j code words. This means that the goal is to find the functions

$$f_{C_j}(v_{C_j}^{(i)}) = \bar{v}_{C_j}^{k} \quad 1 \leq j \leq M, \;\; 1 \leq k \leq K_j, \;\; \forall i$$

where $v_{C_j}^{(i)}$ is a partition of the vector $v^{(i)}$ corresponding to the elements within C_j, and where $\bar{v}_{C_j}^{k}$ is the k^{th} code word for that partition. Note that if $|C_j| = 1 \; \forall j$, then this corresponds to element-wise *scalar* quantization, and if $|C_j| = D$ (implying that $M = 1$), then this corresponds to full *vector* quantization. Anything in between, we will refer to as *subvector* quantization. In this general scheme, any vector element may be clustered with any set of other vector elements. The overall goal is to find the number of clusters M, the clusters themselves $\{C_j\}_{j=1}^{M}$ satisfying the above, the code-book sizes K_j, and the quantization function $\{f_{C_j}(\cdot)\}_{j=1}^{M}$. The above quantities need to be found such that both the total memory and computation required is minimized, and also such that the word error rate (WER) increase (relative to a baseline without quantization) is at a minimum. Because these two minimization criteria are independently optimizable, we report in section 4 results as two-dimensional plots showing WER vs. total space required (equivalently number of bits per parameter). Plots which are both lower and to the left are preferable.

We further distinguish between two quantization styles, *disjoint* vs. *joint* quantization. Disjoint quantization is described above. With *joint* quantization, different clusters are quantized together using the same codebook, meaning that we form the L sets $\{\mathcal{C}_\ell\}_{\ell=1}^{L}$ defining the set of sets $\mathcal{C}_\ell \subseteq \{C_1, C_2, \ldots, C_M\}$ such that $\mathcal{C}_\ell \cap \mathcal{C}_n = \emptyset$ and $\bigcup_\ell \mathcal{C}_\ell = \{C_1, C_2, \ldots, C_M\}$. In this case, the goal is to find the memory-size and WER minimizing functions

$$f_{\mathcal{C}_\ell}(v_{C_j}^{(i)}) = \bar{v}_{\mathcal{C}_\ell}^{k} \;\; \forall C_j \in \mathcal{C}_\ell, \;\; 1 \leq j \leq M, \;\; 1 \leq k \leq K_l, \;\; \forall i$$

such that $|C_i| = |C_j|, \forall C_i, C_j \in \mathcal{C}_\ell$ (i.e., clusters of different size cannot be quantized together), where $\bar{v}_{\mathcal{C}_\ell}^{k}$ is the k^{th} code word for cluster group ℓ.

From the above, we see that there are broadly two separate issues to solve. The first is how to select the number M and set of subvectors $\{C_j\}_{j=1}^{M}$, what we call the *clustering* problem. The second issue is how to perform the quantization once the clustering has been chosen. In this paper, we only address the first issue. However, the quantization algorithm and the quality of the quantization (i.e. the distortion between the codewords and the original vectors) does have an effect on the overall WER. In section 4, we show how the LBG quantization algorithm [7] outperforms LVQ [6] using a particular clustering scheme. All other results are based on the LBG quantization scheme [2].

2.1 Vector, scalar, and composed quantization

Vector and scalar quantization are defined in the previous section. We also define a method we refer to as *composed* quantization, where we first quantize the data vectors using vector quantization, then quantize the scalars of the quantized vectors using joint scalar quantization.

[1]Results in this paper always quantize means and variances separately.

[2]In LBG code words are iteratively split and adjusted to minimize distortion (Euclidian squared distance)

In all our experiments, we compute memory usage as follows: we denote the vector quantization resolution level by $q_{\texttt{vec}}$ and scalar quantization level by $q_{\texttt{sca}}$ then the storage needed for vector quantization is $2 \times (q_{\texttt{vec}} \times N + 2^{q_{\texttt{vec}}} \times D \times 32)$ bits, where the factor 2 is due to the quantization of both the means and variances. The first term in the sum corresponds to the storage required for the indices to the quantized data. The second term corresponds to the size of the codebook. We assume 32 bits are used for unquantized scalars. For joint scalar quantization, the memory usage is $2 \times (q_{\texttt{sca}} \times N \times D + 2^{q_{\texttt{sca}}} \times 32)$ bits. For disjoint scalar quantization, the memory usage is $2 \times (q_{\texttt{sca}} \times N \times D + 2^{q_{\texttt{sca}}} \times D \times 32)$ bits. For composed quantization, $2 \times (q_{\texttt{vec}} \times N + 2^{q_{\texttt{vec}}} \times D \times q_{\texttt{scal}} + 2^{q_{\texttt{scal}}} \times 32)$ bits are required.

2.2 Subvector quantization

Quantization is similar to loss-less compression in that we want to come up with a minimal set of codewords to represent exactly a much larger set of vectors. Compression works because "natural" and human-generated data has a structure far from random i.e. it has redundancy. Entropy, denoted as $H(\cdot)$, is a measure of this randomness and of how predictable a sample of the data is. The lower the entropy the less random the data is and the easier it is to compress. Even more importantly, Shannon proved that the best compression (the minimal number of bits per sample) that can be achieved is bounded below by the entropy. The difference between compression and quantization is that in the latter we allow some amount of distortion between the quantized codewords and the original vectors, which makes it possible to achieve quantization rates below the entropy.

Hereafter, we consider the vector $v^{(i)}$ a sample from a random vector V drawn from some distribution $p(v)$. This is a valid model of the parameters we want to quantize as long as the probability distribution is estimated accurately. Assuming sufficient samples $v^{(i)}$ (i.e., that N is large) and ignoring the codebook size, it can be shown by the law of large numbers that vector quantization (i.e., $M = 1$) is optimal in that it will minimize the overall distortion between the original and the quantized data using a fixed number of bits per element. This can also be shown by the entropy inequality [4],

$$H(V)/D \leq \frac{1}{D}\sum_{j=1}^{M} H(V_{C_j}) \leq \frac{1}{D}\sum_{j=1}^{D} H(V_j) \tag{1}$$

The entropy rate of an arbitrary quantization scheme, $\frac{1}{D}\sum_{j=1}^{M} H(V_{C_j})$ is upper bounded by that of scalar quantization and lower bounded by that of vector quantization.

Vector quantization is optimal even when vector elements are independent. When dependencies do exist, vector quantization becomes even more advantageous since fewer bits are required to jointly encode correlated elements. In our experiments (section 4)we found that it is indeed the case that dependencies exist between different vector elements and that we should thus expect to see benefits in clustering those elements together.

This analysis gives us a potential scheme for optimally quantizing the parameters. We compute $H(V)$, the smallest number of bits per vector we can use without penalty if we were to compress the vectors, and run the best possible quantization algorithm to determine the codebook. However, this is in the ideal case where we assume sufficient data and we do not take codebook size into consideration. In practice, however, these two problems stand out to be crucial issues.

First, given the high dimensionality of the parameter vectors, there is rarely enough data to accurately compute $H(V)$. Second, the cost of storing the codebook tables becomes prohibitive as the number of bits per quantized vector $q_{\texttt{vec}}$ increases. Therefore, an inherent trade-off exists: we prefer large clusters up to the point where the limited amount of data available to perform the multi-dimensional subvector quantization and the size of the tables become inhibiting factors. Subvector quantization, therefore, is an attempt to achieve better than scalar quantization while avoiding the problems mentioned above.

Theoretically speaking, fixing a particular clustering $\{C_j\}_{j=1}^{M}$, the number of bits sufficient to describe the variables without loss of information is given by $\frac{1}{D}\sum_{j=1}^{M} H(V_{C_j})$. We expect that below this amount the WER would begin to increase dramatically. For example, with scalar quantization we would not hope to quantize without error at anything less than $\sum_j H(V_j)/D$ bits per parameter. If the variables in differing clusters are independent, the entropy rate $\frac{1}{D}\sum_{j=1}^{M} H(V_{C_j})$ then reaches its lower bound in equation 1. Therefore we can achieve the fewest number of bits per parameter under this subvector quantization scheme, which we call an optimal clustering.

Designing $\{C_j\}_{j=1}^{M}$ which approximates the optimal clustering, however, is a hopelessly intractable problem. Even in the case where $|C_j| = 2$, finding the optimal clustering has exponential cost. One existing approach therefore is to manually divide the parameters into subsets based on prior knowledge of the vector elements[10]: for example, it might be argued intuitively that the joint entropies $H(\text{MFCCs})$, $H(\text{deltas})$, $H(\text{double deltas})$, $H(\text{log energy})$ will be small. In [2], a greedy algorithm is used to find $M = 13$ clusters that have low entropy.[3] In subsection 2.2.1, we describe an algorithm that subsumes this scheme.

In the case where $C_j = 2 \;\; \forall j$, minimizing entropy is equivalent to maximizing pair-wise mutual information, as seen using [4] the formula $H(V_m, V_n) = H(V_m) + H(V_n) - I(V_m; V_n)$, where $I(V_m; V_n)$ is the mutual information between V_m and V_n. Moreover, standard linear correlation is an approximation to mutual information [4]. Therefore, the more jointly correlated the components of a subvector, the smaller the entropy will be, meaning the distortion between the quantized and unquantized subvector will be minimized.

We can view the D-dimensional parameters as a D-node fully connected weighted undirected graph, where the weight of each edge denotes the mutual information (or correlation) between the corresponding nodes. Clustering therefore can be seen as finding a graph M-partition, where nodes within each partition are as correlated as possible, and nodes between different partitions are as independent as possible.

Based on the above, in this paper we explore various novel data-driven clustering techniques. The basic clustering algorithms are described in the following sections.

2.2.1 Greedy-n Pair

In this first algorithm, which we call *Greedy-n Pair* (where n is a parameter), we perform a tree search with branching factor n. The nodes of the tree are pairs of vector elements (so that $|C_j| = 2 \;\; \forall j$, and $M = D/2$) with the restriction that no two nodes on the path from the root of the tree to a leaf may contain the same element (i.e. each vector element belongs to a unique cluster). The n children of a node are the top n ranked pairs in terms of mutual information between the two corresponding vector elements. The larger n is the more exhaustive the search is, but also the longer the running time.

Given the discussion in the previous section, the goal is to find the path from root to leaf that has the maximum sum of all the mutual information values of the pairs along the path. This algorithm is summarized as follows:

Input: MI values of all possible pairs of vector elements.
Output: Assignment of vector elements to subvectors.

1. Form all possible nodes (pairs of vector elements) and sort them in decreasing order of weight (MI between the two elements of the node).

2. Construct the search tree by placing the top n nodes under the root. For subsequent levels, each node is assigned as children the n nodes that come after it in the ordered list from step 1.

3. Find the path from the root to the leafs that maximizes the sum of the nodes' weights along the path. This can be done recursively in a top-down manner using depth-first search or in a bottom-up fashion as follows: Starting from the leaf level of the tree, consider consecutive subsets of size n (which correspond to children of of different patents) and mark the highest scoring nodes among each subset. Add the weight of those marked nodes to the weight of their parents. Then repeat the same procedure at this higher level. The final cluster choice will be the unique path that contains only marked nodes.

Figure 1 shows an example of a search tree for the algorithm.

Greedy-1 Pair Greedy-1 is a special case ($n = 1$) of the algorithm described above. Here we greedily select the node with maximum MI, assign it to a cluster, then select the next best node consistent with previous choices until $D/2$ nodes are selected. As mentioned above, a similar algorithm has been used in [2], but it uses correlation instead of mutual information to cluster pairs of vector elements.

[3]Actually, they [2] find cluster pairs that are highly correlated, an approximation to low entropy as mentioned in the next paragraph.

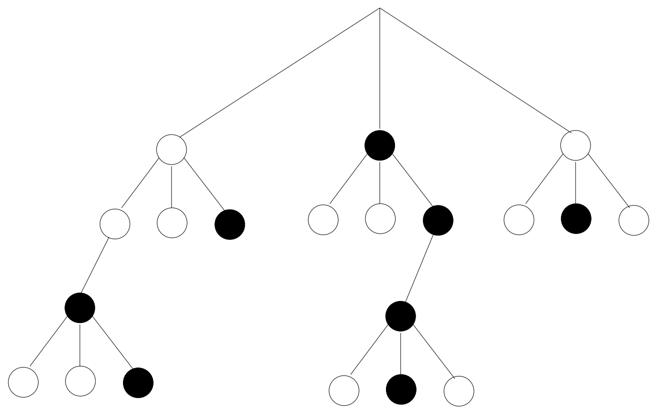

Figure 1: Example of a search tree for the greedy-n-pair algorithm. The tree is complete but for clarity only portions of it are shown. Each node represents a pair of vector elements. The branching factor n is three. Dark nodes are the highest scoring among their siblings. At the leaf level, the score is just the weight of the node. At higher levels, the score is the weight of the node added to the score of its highest scoring child. In the example the middle chain of dark nodes determines the best cluster and it is unique.

Greedy-1 Triplet The Greedy-n Pair algorithm can be generalized to the case where the tree-nodes can have more than two elements ($|C_j| > 2$). In this work, for computational reasons, we consider the less general case when $n = 1$ and $|C_j| = 3$ and we call the technique *Greedy-*1 *Triplet*. In the procedure we implemented, the measure of mutual dependency within elements of a cluster is the average pair-wise mutual information between all pairs of scalar elements. Another more accurate way to evaluate dependency between elements of clusters larger than two is to compare the entropies of the clusters. We discuss this alternative approach in 2.2.2. Other than the different size of the clusters, the selection algorithm is the same as *Greedy-*1 *pair*.

This algorithm can be extended to the more general case, greedy-n m-let where n is the branching factor as before and m is the size of the clusters formed. The measure of dependency here is again the average pairwise mutual information.

2.2.2 Linear Entropy Minimization

The previous schemes require a uniform subvector size (i.e., $|C_i| = |C_j| \quad \forall i \neq j$) even though smaller or larger subvectors might exhibit a higher degree of correlation (and thereby better overall quantization). Another problem with the *Greedy-*1 *Triplet* algorithm is that the mutual information of the clusters is approximated by the sum of pair-wise mutual information values. The advantage of forming clusters of size larger than two is thus lost since we are not able to capture complex dependencies. In the following scheme, we allow clusters with different sizes and calculate the entropy of the cluster exactly by making a linear dependence assumption i.e we assume the probability distribution over vector elements is Gaussian. Then we can calculate the entropy of a subvector V_S of dimension $|S|$ as $H(V_S) = \frac{1}{2}\log((2\pi e)^{|S|} \times det(K))$, where K is the covariance matrix of the Gaussian.

The algorithm proceeds as follows:

Input: Parameter vectors to be quantized and p the maximum cluster size allowed.
Output: Assignment of vector elements to sub-vectors.

1. Calculate the entropies of all sub-vectors of size up to p (there are $\binom{D}{1} + \binom{D}{2} + \cdots + \binom{D}{p}$ possible sub-vectors) and normalize each entropy by the size of the corresponding sub-vector to avoid penalizing large sub-vectors.

2. Rank entropies in increasing order.

3. Iteratively choose the cluster with the lowest entropy and remove all elements in the cluster from further consideration. The cluster assignment is over when there are no elements left.

We will refer to this subvector quantization scheme as *Entropy-min-p* where p is the maximum cluster size allowed. Below we describe a different algorithm that allows varied sized clusters.

2.2.3 Maximum clique quantization

In our *maximum-clique* [4] scheme, we adopt a structural approach in which the dependency graph described in Section 2.2 is pruned so that only a percentage of the edges with weights above some threshold remain. A maximum clique finding algorithm is then applied to the sparse graph. Essentially, it is attempting to minimize $\frac{1}{D}\sum_{j=1}^{M} H(V_{C_j})$ in equation 1, without the constraint that C_j's must be of equal size. When there are two overlapping cliques, the one with the maximum average mutual information is chosen and its elements are removed from the graph.

The algorithm is described below:

Input: MI values of all pairs of vector elements.
Output: Assignment of vector elements to subvectors.

1. Starting from a complete graph (all nodes are connected to each other) where the weight of each edge corresponds to the pair-wise mutual information, eliminate all edges with weights below a chosen threshold.
2. Find the maximum sized clique of the graph. If there is more than one such clique, choose the one with the maximum average mutual information between its nodes.
3. Assign this clique to a subvector and eliminate it from the graph.
4. Repeat 2 and 3 until all nodes are assigned to a subvector.

The search space for the set of thresholds is of course very large but in practice we can choose a threshold by doing a significance test i.e. we compute MI from random data and set the threshold around the highest value observed. Herein we have tried different threshold values that result in clique sizes that tend to perform better.

2.3 Normalized joint quantization

The discussion above assumes disjoint quantization, where each subvector is clustered using a separate codebook. This makes the codebook size a great factor in memory consumption. An alternate scheme is to quantize subvectors of the same size jointly, the motivation being that the codebooks for different subvectors could have much overlap in their value ranges and that more data would be available to perform the quantization on. In the extreme case, where all equal-sized subvectors have the same probability distribution, the subvectors will have identical codebooks. In this case, joint quantization can achieve a distortion as low as that of disjoint quantization while the memory for codebooks is dramatically reduced.

Different subvectors, however, are not necessarily identically distributed. We, therefore, normalize all vector elements to have the same mean and covariance (under the Gaussian assumption, a probability distribution is fully determined by its mean and covariance), apply the joint quantization algorithm, and then convert the quantized vectors back to vectors with the original means and variances. When all the subvectors are scalars, this normalization procedure results in all elements V_j, $j = 1, \cdots, D$ in the vector V being identically distributed. In general, however, component-wise normalization is not sufficient to ensure subvectors V_{C_j}, $j = 1, \cdots, M$ have the same distribution (for that, the off-diagonal elements of the covariance matrices need to be made the same). Nevertheless, the overlap between the ranges of the subvectors increases and in the two cases (scalar subvectors and pairs) in which we used normalization, this scheme proved to work better than quantization without normalization.

The normalization procedure is described below:

1. Calculate the sample mean and variance for each element V_j in vector V.
2. Normalize each element according to its corresponding mean and variance.

[4]Cliques are often defined as a maximal complete set, In this work we call it the maximum clique even though this might be redundant

3. Quantize the normalized parameters. (Here we applied normalized joint quantization scheme for both scalar and greedy-1-pair.)

4. "Denormalize" each element according to its corresponding mean and variance.

There is a small issue when converting quantized normalized variances back to their original value ranges in that these variances can sometimes take negative values. We solve this problem by clamping such negative variances to zero (or a very small value), which works well since, for a variance to be assigned a negative codeword, it must have been close to zero in the first place.

3 DATABASE

In all experiments reported in this work, we use NYNEX PHONEBOOK, a phonetically-rich, isolated-word, telephone-speech database[9]. Speech data is represented using 12 MFCCs plus c_0 and their deltas resulting in a $d = 26$ element feature vector every 10ms. The training and test sets are as defined in [1]. Test words do not occur in the training vocabulary, so test word models are constructed using phone models learned during training. Strictly left-to-right transition matrices were used except for an optional beginning and ending silence model. Four states per phone were used leading to a total of 165 hidden states, using the dictionary contained with the PHONEBOOK distribution.

PHONEBOOK is an ideal database to use for this study because its contains a variety of isolated words, which, we think, will be the most likely form of speech input using handhelds at least in the near term.

In our results, we quantize only the means and variances of Gaussian distributions which are used to model the state output probabilities in a continuous density HMM. Mixture coefficients are left unquantized. Quantizing them neither achieves significant memory savings since they account for less than 5% of the number of means or variances nor does such an operation affect WER in a significant way. There are a total of 1900 mean and variance vectors, leading to $1900 \times 26 \times 2 = 98800$ 32-bit scalars for both the means and variances.

Results reported here were obtained on 150-word test sets, a size we think would be typical for the kind of applications that use few memory and computational resources. We have also verified that the results hold for 300 and 600 word sets.

4 RESULTS

In this section, we evaluate the various clustering methods that were described in previous sections.

4.1 LBG versus LVQ

We first compare two quantization algorithms under the same clustering scheme, joint scalar quantization. Figure 2 shows that the LBG quantization algorithm yields better recognition accuracy than the LVQ algorithm. However, the WER-storage curves of both algorithms still have the same trend, they both show a sharp decrease in WER at a certain quantization level (around five bits per parameter, taking into account the table size).

4.2 Vector, composed, joint scalar, and disjoint scalar

Figure 3 shows a comparison between vector, composed, scalar, and disjoint scalar clustering methods. The single cross at the right of the plot shows baseline performance, with no quantization (meaning 32-bits per parameter, and a memory cost that does not require a table). As can be seen, the scalar quantization schemes (both joint (non-normalized) and disjoint) perform significantly better than either the vector or the composed schemes. Joint scalar quantization uses only 15.7% of the baseline memory, and keeps the WER as low as 2.55% (only a 5.0% relative increase over the baseline). Composed quantization alleviates the table size problem which penalizes vector quantization, but still does not beat scalar quantization.

It is also worth noting that although disjoint scalar quantization would seem to require more memory than joint scalar quantization given its additional table table storage, it achieves a WER of 2.46% with just 13.3% of the baseline memory.

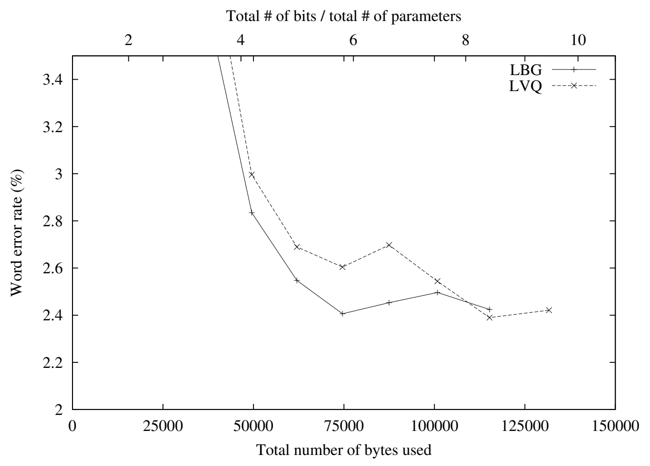

Figure 2: Comparison between the effect on speech recognition WER of two different quantization algorithms, LBG and LVQ. Note that the bottom and top x-axis are labeled differently: the bottom axis is the total number of bytes required (both for indices and tables), the top axis is the total number of bits used for the quantization scheme, including table memory, divided by the total number of ASR-system parameters (Gaussian mean and covariance scalar values).

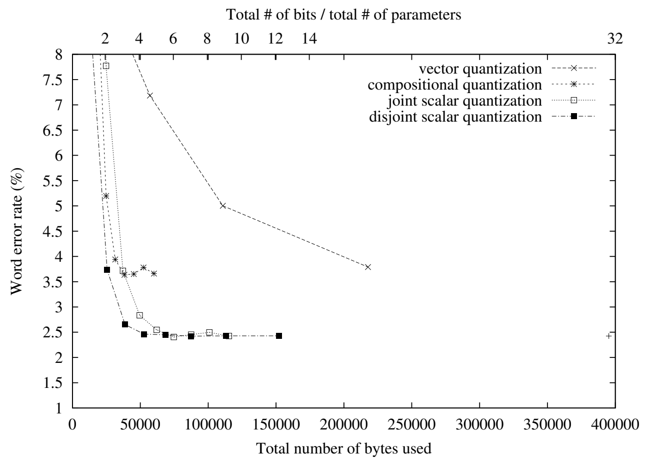

Figure 3: Comparison between vector, composed, joint scalar and disjoint scalar quantization.

4.3 Subvector quantization

Even though, theoretically, subvector quantization should compress parameter vectors better than scalar quantization regardless of whether any correlation exists between vector elements, it is certainly desirable to have strong correlation to improve quantization even further. Figure 4 shows that there are indeed dependencies between vector element pairs, as evidenced by positive mutual information values that are larger than those derived from random data. Moreover, MI values from random data are very close to zero as expected when there is enough data, which increases our confidence–but does not guarantee–that real data MI values are also accurately estimated.

Figure 5 shows a comparison of disjoint scalar, Greedy-1 Pair and a random procedure (cluster pairs are chosen randomly). Clearly the Greedy-1 Pair subvector selection procedure is superior to random and there is an advantage in clustering correlated elements together. However the scheme still does not outperform the simpler disjoint scalar quantization. There are several reasons this might be so. First of all, the greedy procedure is a heuristic, which does not ensure we select the best subvectors. Second, estimating the mutual information relies on having enough data. We have tried using correlation as a substitute for MI (see figure 7), since it does not require as much data to be estimated accurately. Third, comparing the number of possible codewords that can be formed using two bits per scalar value, for example, the disjoint scheme yields $(2^2)^{26}$ possible codewords while the greedy-1 pair only $(2^2)^{13}$. Of course,

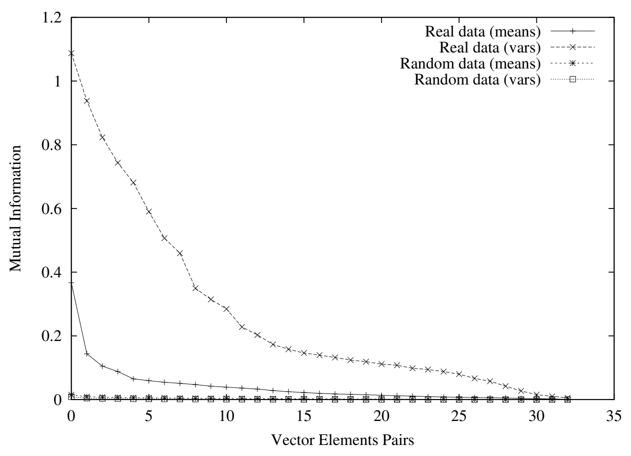

Figure 4: Mutual information values of ordered pairs of vector elements for real and random data (for both means and variances). Random data is generated according to a Gaussian distribution with the same parameters as real data. For clarity, only every tenth of the $\binom{26}{2}$ pairs are shown.

the greedy scheme uses less memory for storing the indices but that apparently is not enough to offset the loss in recognition accuracy.

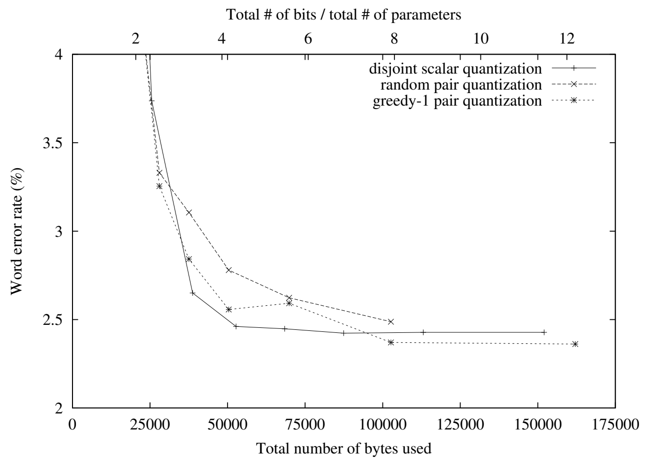

Figure 5: Comparison of greedy-1, random, and disjoint scalar.

Figure 6 compares the greedy-n pair procedure, with $n = 1$ and $n = 8$. for $n \in \{3, 4, 5, 6, 7, 8\}$ the clustering results were exactly the same, which is why we report only these two cases. As can be seen, there does not seem to be a clear advantage of greedy-8 pair over greedy-1 pair. If we compare the sums of the entropies in each case ($n \in \{1, \cdots, 8\}$), we find that there is a very small entropy reduction when going from $n = 1$ to $n = 8$. While it might be that greedy-n pair for $n > 8$ would achieve a better clustering, the computational cost of the clustering algorithm quickly becomes prohibitive.

Figure 7 shows the results of greedy-1 pair computed using two different forms of mutual information approximations. The first way uses a mixture of Gaussians to estimate the joint density of the pair, and then uses that joint density to compute the mutual information. The second way simply assumes that the two random variables are jointly Gaussian, and computes the resulting MI analytically [4]. Note that the second way is equivalent to computing simple linear correlation. It is safe to say linear correlation is sufficient, probably because the amount of data ($n = 1900$) is not large enough to produce a reliable estimate of more accurate mutual-information approximations.

Figure 8 compares disjoint scalar, greedy-1 pair, and greedy-1 triplet. The results show that using this heuristic, clusters of size three $|c_j| = 3$ does not perform better than simple disjoint scalar quantization. Again, the fact that we

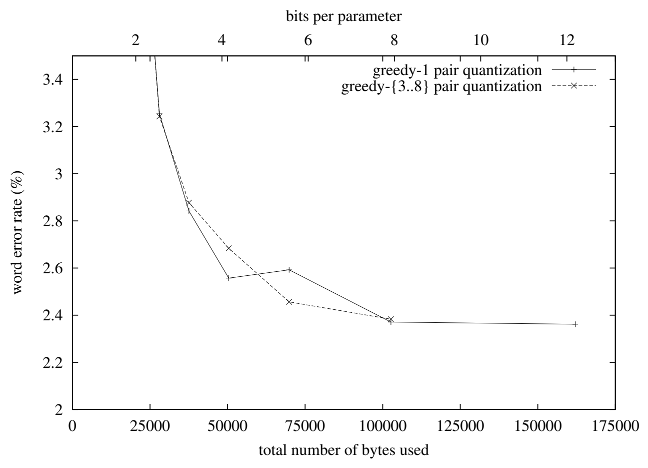

Figure 6: Comparison of greedy-1 and greedy-3(4,5,6,7,8) pair quantization.

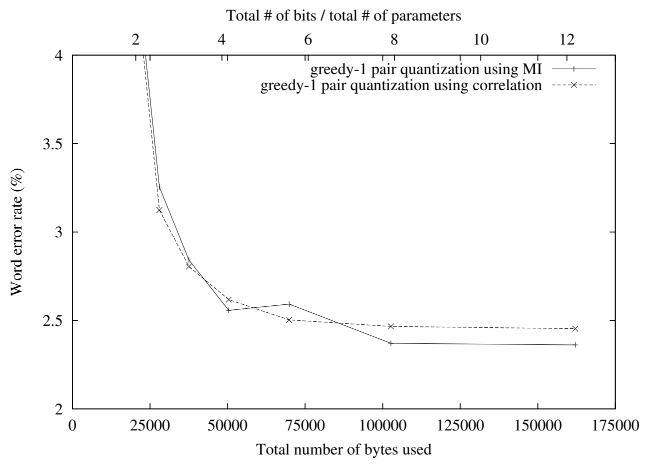

Figure 7: Comparison between pairwise quantization using MI and correlation.

are using pair-wise MI to approximate the MI of the triplets partly accounts for this. Also the issue of data sparsity (small n) makes it hard to estimate MI using a complex probability distribution model (a mixture of Gaussians in our case).

Figure 9, however, shows that even the simpler to estimate linear entropy does not improve over greedy-1-pair or the disjoint scalar schemes. We can notice that as the size of clusters increases (from entropy-min-2 to entropy-min-6) the WER-memory curve is shifted to the left but at the same time to the top i.e. savings in memory are offset by a loss in recognition accuracy. The reason for this is that, in general, with larger clusters, more bits per cluster need to be used to encode a given number of different codewords compared to smaller clusters. And even though the average number of bits per vector element is actually smaller for larger clusters (remember, vector quantization is the most efficient when we do not count the table size), the table size–an exponential term–quickly swamps clusterings that use large clusters.

Figure 10 compares different thresholds for the maximum clique strategy. The thresholds we choose in this experiment are 4% and 8%, meaning we keep either 4% or 8% of the highest weight edges in the graph. Once again, the performance is no better than disjoint scalar quantization. In further experiments (not shown in the plot), we find that differences are negligible with a threshold ranging between 4% and 15%, and that quantization gets worse when the threshold is increased further.

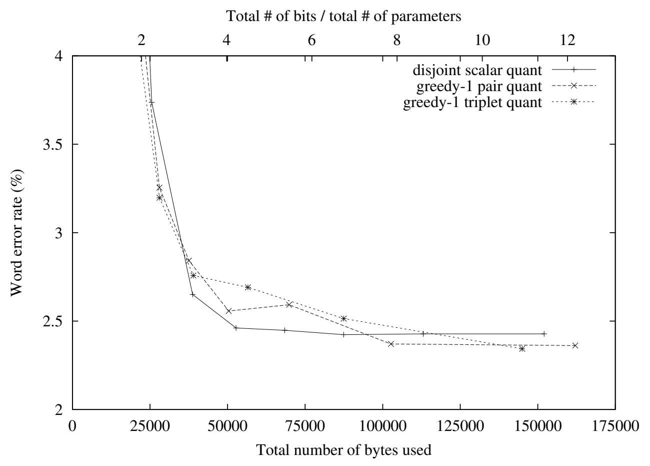

Figure 8: Comparison between one, two and three dimensional subvector quantization.

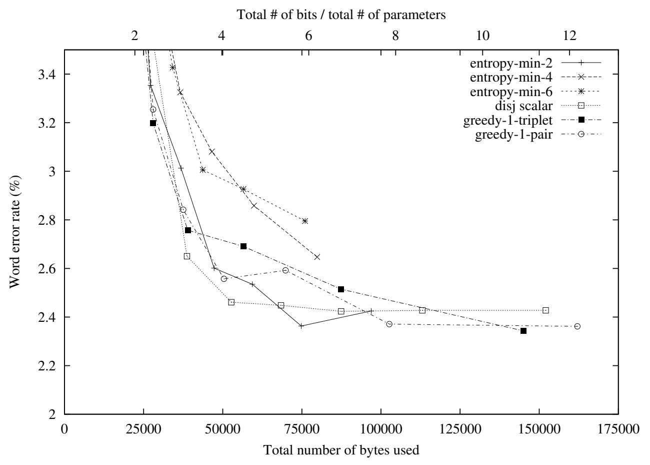

Figure 9: subvector quantization using linear entropy minimization. Entropy-min-3 and entropy-min-5 are not shown for clarity but they follow the same trend as entropy-min2,4,6

4.4 Normalized joint quantization

Lastly, in figure 11, we compare normalized joint vs. disjoint quantization for two different clustering methods (scalar and greedy-1 pair). The joint quantization here is pre-processed by normalization in both pairwise and element-wise cases.

The joint schemes, normalized joint scalar and normalized joint greedy-1 pair, do slightly better than disjoint scalar and disjoint greedy-1 pair respectively. For example the joint scalar scheme reaches the baseline WER (2.42%) with only 12.5% of the baseline memory, while the disjoint scheme achieves an WER of 2.49% with 13.3% of the baseline memory. Given the small difference in performance and that normalized joint schemes involve two extra steps, normalization, which is done offline and the conversion to the original range of values, which has to be done online [5], the disjoint schemes seem to be good choices.

With regards to the difference between normalized schemes and their non-normalized counterparts, the former perform much better as is expected. Note that our normalization is performed at the scalar level even in the case where subvectors are pairs of vector elements. Implicit in this approach is that we assume independence of the subvector elements, which of course is not true (in the contrary pairs were chosen to maximize dependence between the elements). Nevertheless, the normalized greedy-1 pair outperformed the corresponding non-normalized scheme.

[5]It is is possible to perform the conversion offline at the cost of more memory usage, since we would need to store as many codebooks as there subvectors instead of just one codebook

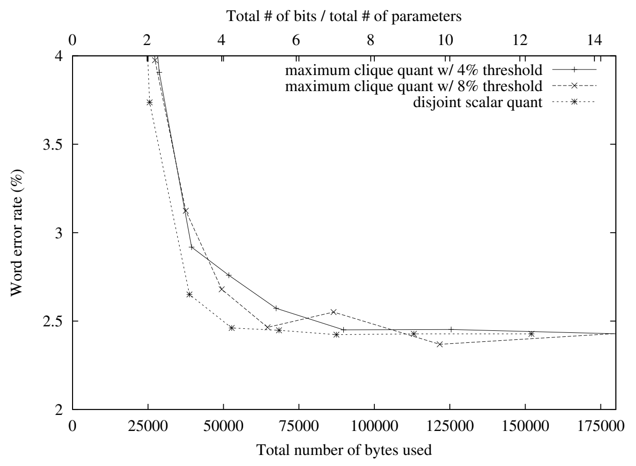

Figure 10: Threshold-based maximum clique quantization.

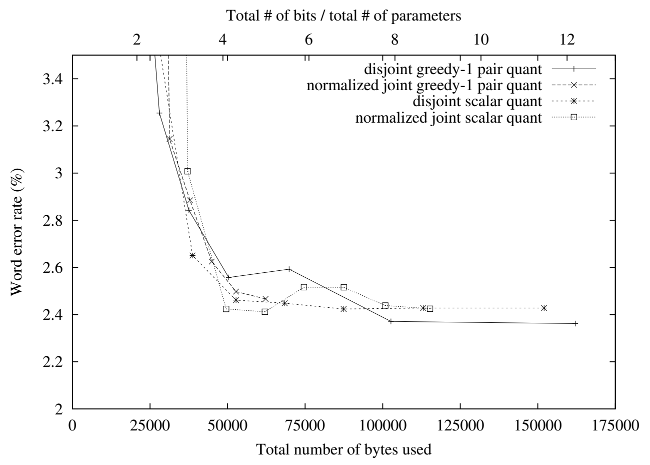

Figure 11: Comparison between joint and disjoint quantization schemes.

Figure 12 shows the comparison between the normalized and the non-normalized schemes above.

5 CONCLUSION

The results above evaluate a number of novel methods for producing subvector-based parameter quantization in Gaussian-mixture HMM-based ASR systems. We find that three methodologies are the overall best in their attempt to reduce memory: disjoint scalar, normalized joint scalar and normalized joint greedy-1-pair quantization. They do better than more elaborate heuristics. With less than a seventh of the parameters storage, a near baseline WER is achieved.

Even though the disjoint scalar scheme requires more storage for its separate codebooks, its memory/WER curve is close to the best one achieved. Moreover it does not require the normalization/re-conversion step in the normalized schemes. It is surprising that a disjoint scheme would do well at all because of the table storage, but it becomes less surprising when we look at the added flexibility to form codewords when separate codebooks are used. Joint schemes are constrained to share the same codebook among subvectors and therefore for a fixed number of bits per codeword, they encode fewer combinations of codewords than a disjoint scheme does.

An issue that we have not investigated in this work is the interaction between the clustering schemes and the quantization algorithms. As already mentioned that we have used a single quantization algorithm, which works best with the two clustering schemes we have tried. It is possible that different combined clustering-quantization schemes

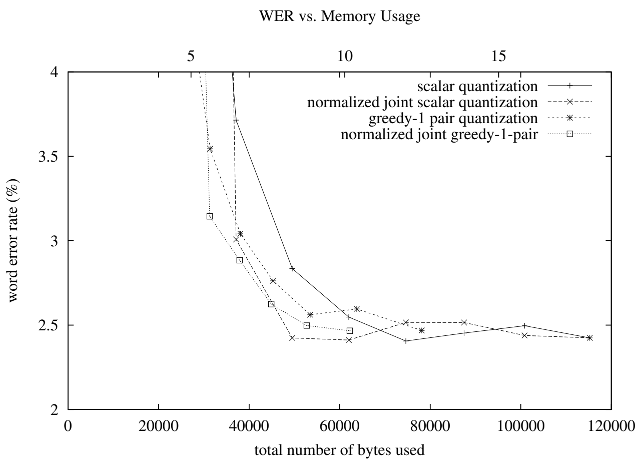

Figure 12: Comparison between normalized and non-normalized schemes.

would perform better, although we do not expect the improvements to be very important given the difficulty to improve upon the simple schemes presented in this paper.

Acknowledgments

We would like to thank NSF for funding this project under Award-0086032.

References

[1] J.A. Bilmes. Buried Markov models for speech recognition. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, Phoenix, AZ, March 1999.

[2] Enrico Bocchieri and Brian Mak. Subspace distribution clustering for continuous observation density hidden markov models. In *Proc. Eurospeech '97*, pages 107–110, Rhodes, Greece, 1997.

[3] L. Comerford, D. Frank, P. Gopalakrishnan, R. Gopinath, and J. Sedivy. The IBM personal speech assistant. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, Salt Lake City, Utah, 2001.

[4] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[5] X. Huang, A. Acero, C. Chelba, L. Deng, D. Duchene, H. Hon, and et al. Mipad: A next generation pda prototype. In *Proc. Int. Conf. on Spoken Language Processing*, Beijing, China, November 2000.

[6] T. Kohonen, J. Hynninen, J. Kangas, J. Laaksonen, and K. Torkkola. Lvq pak: The learning vector quantization program package, 1995.

[7] Y. Linde, A. Buzo, and R.M. Gray. An algorithm for vector quantizer. *IEEE transactions on communications*, COM-28(1):84–95, 1980.

[8] G. Moyers. A handy way to interact. *Speech Technology*, pages 14–17, July/August 2001.

[9] J. Pitrelli, C. Fong, S.H. Wong, J.R. Spitz, and H.C. Lueng. PhoneBook: A phonetically-rich isolated-word telephone-speech database. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, 1995.

[10] M. Ravishankar, R. Bisiani, and E. Thayer. Subvector clustering to improve memory and speed performance of acoustic likelihood computation. In *Proceedings of Eurospeech*, pages 151–154, Rhodes, Greece, 1997.

[11] Kaushik Roy and Mark C. Johnson. Software design for low power. In *Low Power Design in Deep Submicron Electronics. Proceedings of the NATO Advanced Study Institute*, pages 433–460, Dordrecht, Netherlands, 1997. Kluwer Academic Publishers.

[12] M. Vasilache. Speech recognition using HMMs with quantized parameters. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, 1999.