VERY FAST TREE-STRUCTURED VECTOR QUANTIZATION

Todd K. Moon and Christian B. Peel and Scott Budge

Department of Electrical and Computer Engineering
Utah State University Logan, UT 84322-4120
Todd.Moon@ece.usu.edu, chris.peel@ieee.org, Scott.Budge@ece.usu.edu

ABSTRACT

Very fast tree-structured vector quantization employs scalar quantization decisions at each level, but chooses the dimension on which to quantize based on the coordinate direction of maximum variance. Because the quantization is scalar, searches are no more complex than scalar quantization — providing significant improvement in complexity over full-searched or even tree-structured vector quantization — but the method preserves the shape and memory advantages of conventional vector quantization. However, the space filling advantage of VQ is forfeited, since each Voronoi cell is a rectangular cuboid.

1. INTRODUCTION

Tree-structured vector quantization (TSVQ) is widely known and appreciated for its relatively fast search properties [1]. Algorithms for efficiently training of tree-structured codebooks are investigated in [2, 3]. In this paper, we examine empirically the performance of a method of suboptimal TSVQ that has search complexity lower even than TSVQ, by a factor that is the dimension of the codevector. The quantization complexity is comparable to scalar quantization at the same rate. For example, for binary trees, the search complexity is *one compare per output bit.* As we demonstrate, this method provides many of the advantages of traditional vector quantization [4, 5]: it takes advantage of both linear and nonlinear correlation and shapes the quantizer according to the shape of the training data. However, because all the the Voronoi regions are rectangular cuboids, it lacks the space-filling advantage of other vector quantization techniques. While the quantization is suboptimal, there may be circumstances in which the tradeoff in performance and low computational complexity favors this approach.

The concept of the tree-structured VQ described here stems from the classification and regression trees described in [6], in which data are successively partitioned and thereupon treated independently. In [7], a similar approach to forming the decision boundaries based on local splitting rules is presented, but the analysis there is directed toward asymptotic quantization performance. However, in these sources, performance on trained data sets and the very fast nature of the encoding algorithm are not described.

2. VERY FAST VECTOR QUANTIZATION

Conventional TSVQ consists of an l-level Q-ary tree, where $Q = 2^b$, and b is the number of bits per level. When full-depth quantization is performed on the tree, an equivalent codebook of $N = 2^{lb}$ vectors is available, at a search cost of only $O(lb)$ search operations per vector, in comparison to $O(2^{lb})$ searches for full-search vector quantization (FSVQ). TSVQ thus has considerable computational advantage in comparison to FSVQ. However, extant codebook design algorithms for TSVQ lead to inferior distortion performance of TSVQ compared to FSVQ. In conventional TSVQ, each of the lb comparisons in a full-depth search is actually a *vector* comparison. For d-dimensional vectors, there are thus actually $O(lbd)$ operations necessary to search the tree. (If early abort vector comparisons are used, then the complexity might be somewhat less on average.) Very-fast tree-structured vector quantization (VFTSVQ) is accomplished by designing a codebook so that vector comparisons are not necessary, so the searching is accomplished d times faster than for TSVQ: the computational complexity is the same as for scalar quantization with the same number of bits. Just as TSVQ is faster but has more distortion than FSVQ, so VFTSVQ is faster than TSVQ, but being suboptimal has more distortion.

Given the increase in distortion, it is worthwhile to ask what role VFTSVQ might play in modern data compression systems. One potential area is in conjunction with transform-based compression on processing-limited platforms. If a significant fraction of the processor's cycles are employed in computing transforms or in other pre-processing tasks, expensive quantization may not be an option. In such a circumstance, VFTSVQ might provide some quantization performance not available using scalar quantization, but without significant computational overhead.

In the remainder of this paper, we focus primarily on binary trees ($b = 1$). Extension to wider trees is straightforward. In binary VFTSVQ, it is desired to determine a single quantization level so that under scalar quantization — quantization with respect to a single coordinate element of a vector — the quantized representation has minimum distortion. Let $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$ be a set of T training vectors of d-dimensional data, and let $\mathbf{x}_i(j)$ be the jth element of $\mathbf{x}_i$. We define a scalar projection set in coordinate δ by

$$\mathcal{X}_\delta = \{\mathbf{x}_1(\delta), \mathbf{x}_2(\delta), \ldots, \mathbf{x}_T(\delta)\}.$$

In VFTSVQ, this projected data set is used to determine

a quantizer satisfying the Lloyd conditions [8], yielding a quantizer $(\tau, \hat{\mathbf{x}}_0, \hat{\mathbf{x}}_1)$, where τ is the optimal (Lloyd) threshold designed for the data $\mathcal{X}_\delta$, and $\hat{\mathbf{x}}_i$ are centroids of the regions defined by

$$\{\mathbf{x} \in \mathcal{X}\colon \mathbf{x}(\delta) < \tau\} \qquad \text{and} \qquad \{\mathbf{x} \in \mathcal{X}\colon \mathbf{x}(\delta) \geq \tau\}.$$

Since the quantizer threshold τ is designed with respect to the scalar data $\mathcal{X}_\delta$, the quantizer is as fast as scalar quantization. And since the centroids are chosen with respect to the vector data the quantizer offers the shape and memory advantages.

At each stage of the algorithm, we must determine which coordinate direction δ to project onto and split. To minimize the distortion, we will split the data set in the direction of maximum variance. We thus choose

$$\delta = \arg \max_{1 \leq i \leq d} \text{var}(\mathcal{X}_i),$$

where $\text{var}(\cdot)$ computes the variance of the set argument.

The training algorithm operates recursively. Given a set of data, a centroid, splitting dimension and threshold are selected. The data are partitioned according to the threshold and are then used to train the next level of the tree, until the maximum desired tree level is reached. If any empty cell arises in this training procedure it may be treated as follows. At each level, the boundaries of the decoding region are saved. When a cell is empty, the geometric midpoint of the decoding region is selected as the centroid, and if additional levels of the tree are needed, the cell is split in the direction of maximum cell width.

3. DEMONSTRATIONS OF VFTSVQ

In this section we provide several examples illustrating the performance of binary VFTSVQ.

Example 1 A large number of correlated 2-dimensional Gaussian data points were generated, and a VFTSVQ tree was produced. Figure 1 shows the decision boundaries (the dashed lines) — demonstrating both the splitting dimension and threshold — as well as the centroids of the regions produced (the $\times$s) as the number of levels in the tree increases from one to six. From this example we may observe that (1) The large structure of the data is captured, so that (in this example) centroids tend to fall along the major axis of the data; and (2) Centroids tend to cluster more in regions of high probability, being more dense near the middle and sparser near the edge. Thus the shape property of VQ is retained. On the other hand, the Voronoi regions are rectangular, so that this method does not exploit the potential shape advantage. □

Example 2 One of the frequently stated reasons for using VQ is to avoid putting centroids where there is no data. In this example, we demonstrate that VFTSVQ can be successful at placing centroids where there are representative data. Figure 2 shows 1000 points of two-dimensional data, where the data are uniformly generated over $(-2, 2)$, except for the inner square from $(-1, 1)$ in each dimension. The figure also shows the decision boundaries and centroids of a six-level binary VFTSVQ coder. While the decision boundaries cross the empty region (as they must), the centroids are uniformly distributed among the data points. □

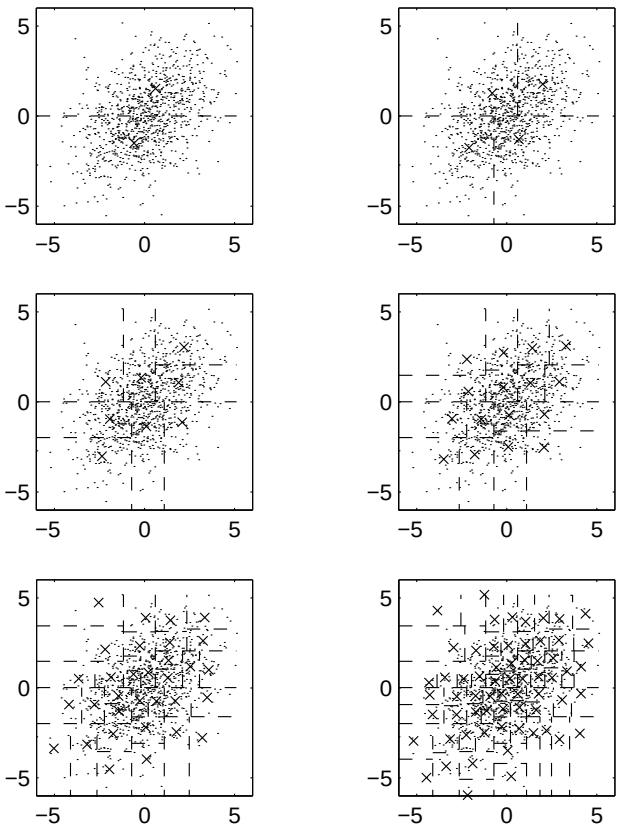

Figure 1: VFTSVQ demonstration on correlated Gaussian data for one to six levels

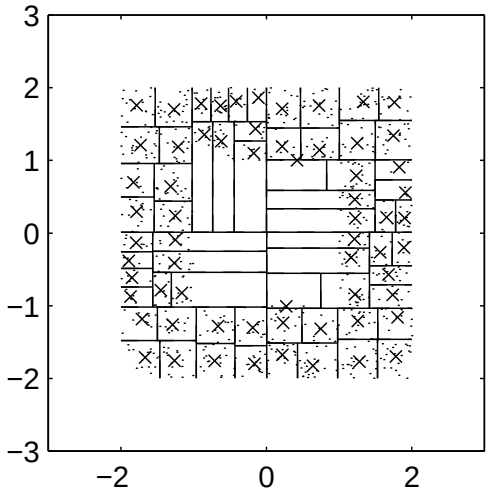

Figure 2: VFTSVQ demonstration of data with gaps

Example 3 We now compare the performance of VFTSVQ with TSVQ and FSVQ on actual image data. Codebooks of up to 8 bits/vector were developed on a large training set obtained by removing the mean from several images and dividing them up into vectors. Vector sizes of 4×2, 4×4, 8×4 and 8×8 were examined. Codebooks for the VFTSVQ, TSVQ, and FSVQ were separately generated and tested. The performance is plotted as PSNR in dB vs. bits/pixel. With the image size M pixels by N pixels, let $\mathbf{i}$ be an MN-dimensional vector representing entire the original image, and let $\hat{\mathbf{i}}$ represent the coded image. Then the PSNR is computed as

$$\mathrm{PSNR} = 10 \log_{10} \frac{255^2}{\|\mathbf{i} - \hat{\mathbf{i}}\|_2^2 / (MN)}$$

In each case, the rate is computed by assuming that 8 bits are used to send the mean of each coded vector, so the number of bits per vector for an l-bit codebook is $(8 + l)$, and the bits per pixel is $(8 + l)/(mn)$, where the vector dimension is $m \times n$.

Figure 3 illustrates the PSNR for the `woman` image . (The final paper will demonstrate results for other images as well, which don't fit here.) As expected, the performance of FSVQ is superior to that of TSVQ, which in turn is superior to that of VFTSVQ. On the other hand, the computational savings is tremendous. For example, for an 8×8 FSVQ coded to eight bits, $O(64 \cdot 256) = O(16384)$ operations must be performed for each each bits, while for TSVQ $O(64 \cdot 8) = O(512)$ operations must be performed. VFTSVQ requires only $O(8)$ operations.

A general observation about this comparison is that the lower the bit rate (for a given vector size) the closer VFTSVQ is to TSVQ. This suggests that VFTSVQ might be best used in high compression scenarios. □

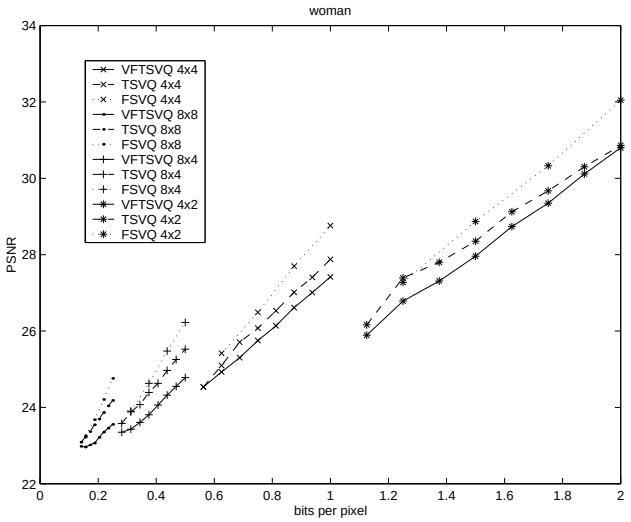

Figure 3: Comparison of VFTSVQ, TSVQ, and FSVQ for the `woman` image

4. SUMMARY

We have described the training, encoding, and decoding algorithms for VFTSVQ, and provided empirical evidence that the method provides space-filling and memory advantages of other forms of VQ. However, the shape advantage is not provided. The penalty for using VFTSVQ is lowest for shallow trees (low bit rates).

5. REFERENCES

[1] A. Buzo, A. Gray, R. Gray, and J. Markel, "Speech coding based upon vector quantization," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 28, pp. 562–574, 1980.

[2] M. Balakrishanan, W. Pearlman, and L. Lu, "Variable rate tree-structured vector quantizers," *IEEE Trans. Information Theory*, vol. 41, pp. 971–930, July 1995.

[3] E. Riskin and R. Gray, "A greedy tree growing algorithm for the design of variable rate vector quantizers," *IEEE Trans. Signal Processing*, vol. 39, pp. 2500–2507, 1991.

[4] T. D. Lookabaugh and R. M. Gray, "High–resolution quantization theory and the vector quantizer advantage," *IEEE Trans. IT*, vol. 35, pp. 1020–1033, September 1989.

[5] J. Makhoul, S. Roucos, and H. Gish, "Vector quantization in speech coding," *Proc. IEEE*, vol. 73, pp. 1551–1588, Nov 1985.

[6] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees.* Belmont: CA: Wadsworth, 1984.

[7] A. B. Nobel, "Recursive partition to reduce distortion," *IEEE Trans. Information Theory*, vol. 43, pp. 1122–1133, July 1997.

[8] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression.* Dordrecht, Netherlands: Kluwer Academic Publishers, 1992.