ONE-PASS VECTOR QUANTIZER DESIGN BY SEQUENTIAL PRUNING OF THE TRAINING DATA

Qi Li and Peter F. Swaszek

Department of Electrical and Computer Engineering
University of Rhode Island, Kingston, RI 02881
Email: qli@ele.uri.edu swaszek@ele.uri.edu

ABSTRACT

A one-pass vector quantizer design algorithm is presented. The algorithm sequentially selects a subset of the training vectors (in a high density area of the training data space) and computes a VQ code vector. Next, a sphere is constructed about the code vector whose radius is determined such that the encoding error for points within the sphere is acceptable. Finally, the data within the sphere is then pruned (deleted) from the training data set. This procedure continues on the remainder of the training set until the desired number of code vectors are located. This one-pass VQ design algorithm is compared with several benchmark results for uncorrelated Gaussian, correlated Gaussian, and Laplace sources; its performance is seen to be as good or better than the benchmark results. Further, the one-pass algorithm needs only slightly more computation than a single iteration of the LBG algorithm.

1. INTRODUCTION

Vector quantization (VQ) has great potential in data compression because of its promising compression ratio and conceptually simple implementation. VQ is also the front end for many classification and pattern recognition problems. To achieve high performance, high computational complexity in both the VQ encoding and codebook design are required. The methods most often employed for designing the codebook, such as the generalized Lloyd or Linde-Buzo-Gray (LBG) algorithm [1], are iterative algorithms and require a large amount of CPU time to converge to a (locally optimum) solution. To speed up the design procedure, we developed a one-pass VQ codebook design algorithm.

2. THE ONE-PASS VQ ALGORITHM

The one-pass algorithm is based on a sequential statistical analysis of the local characteristics of the training data and a sequential pruning technique. It is motivated from a constructive neural network design concept for classification and pattern recognition [2]-[5].

The basic steps of the one-pass VQ design algorithm are illustrated in Figure 1 which shows how one code vector is determined for a two-dimensional training data set (the algorithm is developed and implemented in N dimensions). We refer to this figure in the following discussion.

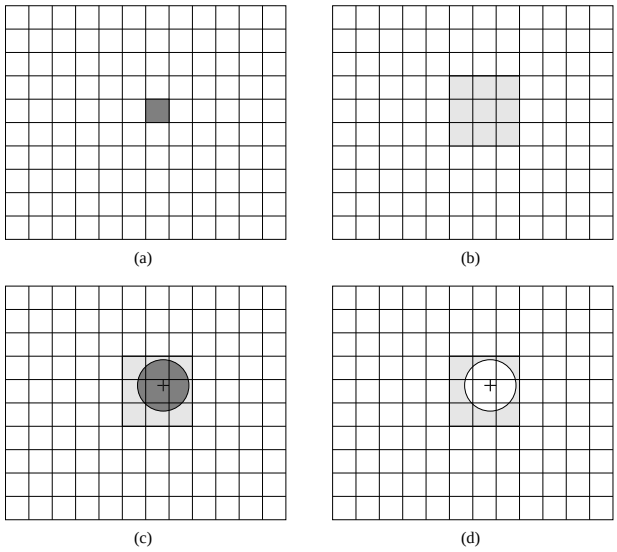

Figure 1: The method to determine one code vector: (a) select the highest density cell; (b) examine a group of cells around the selected one; (c) estimate the center of the data subset; (d) cut a "hole" in the training data set.

2.1. Design Specifications

The training data set X is assumed to consist of M N-dimensional training vectors. The total number of regions R is determined by the desired bit-rate. Let D represent the *maximum allowable distortion*

$$||x_m - y_c||^2 \leq D \tag{1}$$

where $x_m \in X$ is a data vector and y_c is the nearest code vector to x_m. The value for D can be determined either from the application (such as from human vision experiments, etc.) or estimated from the required data compression ratio.

2.2. Source Histogram

The one-pass design first partitions the input space by a cubic lattice to compute a histogram. Each histogram cell is a N-dimensional hypercube. (In general, we could allow different side lengths for the cells.) The number of cells in

each dimension is calculated based on the range of the data in that dimension and the allowable maximal distortion D. The employed method for calculating the number of cells in the jth-dimension, L_j, is

$$L_j = \lceil \frac{x_{j,\max} - x_{j,\min}}{2D/3} \rceil \quad (2)$$

where $x_{j,\max}$ and $x_{j,\min}$ are, respectively, the maximal and minimal values of X in dimension j. The term $2D/3$ is the recommended size of the cell in that dimension. The probability mass function $f(k_1, k_2, \ldots, k_N)$ with $k_i \in \{1, 2 \ldots L_i\}$ is determined by counting the number of training data vectors within each cell.

The sequential design algorithm starts from that cell which currently has the largest number of training vectors (the maximum of $f(\cdots)$ over the k_i). In Figure 1(a) we assume that the highlighted region is that cell. Then, as shown in Figure 1(b), a group of contiguous cells around the selected one is chosen. This region $X_s \subset X$ (highlighted) will be used to locate a VQ code vector.

2.3. Locating a Code Vector

Two algorithms have been employed in locating a code vector: a principal component method and a direct median method. For the results presented here, the medians of X_s in each of the N dimensions are computed as the components of the code vector. Medians are employed to provide some robustness with respect to variation in the data subset X_s. The median is marked in Figure 1(c) by the "+" symbol.

2.4. Pruning the Training Data

Once the code vector is located, the next step is to determine a sphere around it as shown in Figure 1(c). (We note that for classification applications it could be an ellipse.) One way to determine the size of the sphere is to estimate the maximal number of data vectors which will be included inside the sphere. The set of vectors within the sphere X_c is a subset of X_s, $X_c \subset X_s$. To determine a sphere for the cth code vector, the total number of data vectors T^c within the sphere is estimated by

$$T^c = W^c \frac{M^c}{R+1-c} \quad (3)$$

where M^c is the total number of data vectors in the current training data set ($M^1 = M$ and $M^c < M$ when $c > 1$) and $R + 1 - c$ is the number of code vectors which have yet to be located. The term W^c is a variable weight

$$W^c = W^{c-1} - \triangle W \quad (4)$$

where $\triangle W$ is the change of the weight variable between each of the algorithm's R cycles. The weight is employed to keep the individual spheres from growing too large. From the experience of the design examples in this paper, we note that the resulting performance is not very sensitive to either W^1 or $\triangle W$.

After the number of vectors of the sphere X_c is determined, the data subset X_c is pruned from the current training data set. As shown in Figure 1(d) a "hole" is cut and the data vectors within the hole are pruned. The entries of the mass function $f(\cdots)$ associated with the highlighted cells are then updated. The design for the next code vector starts by selecting that cell which currently contains the largest number of training vectors.

Due to the nature of the design procedure it can be imagined that the diameters of the spheres might get larger and larger while the design is in progress. To alleviate this, each sphere's diameter is limited to $2D$. This value is chosen to avoid the situation in which one sphere entirely becomes a subset of another sphere (some overlap is common).

2.5. Updating the Designed Code Vectors Once

After locating all R code vectors and cutting the R "holes" in X there often remains a residual data set $X_l \subset X$ from the training set. The last step of our one-pass design is to update the designed code vectors by calculating the centroids of the nearest vectors around each code vector. This is equal to one iteration of the LBG algorithm.

3. CODEBOOK DESIGN EXAMPLES

In this section the one-pass algorithm is tested by designing codebooks for uncorrelated Gaussian, correlated Gaussian, and independent Laplace sources since many benchmark results on these sources have been reported [6, 7, 8, 9, 10, 11, 12, 13, 14].

To facilitate the comparison of CPU time and Flops (floating point operations) from different systems we use the well-known LBG algorithm as a common measurement and define the following two kinds of speed-up

$$\text{Speed-up-in-time} = \frac{\text{CPU time for LBG}}{\text{CPU time for algorithm}} \quad (5)$$

$$\text{Speed-up-in-flops} = \frac{\text{Flops for LBG}}{\text{Flops for algorithm}}. \quad (6)$$

Since we use a high-level, interpreted language for simulation on a multi-user system, we prefer the Speed-up-in-flops as a comparison measurement.

3.1. Uncorrelated Gaussian Source

3.1.1. Two dimensional Gaussian source

In this example, the data vectors are from a zero-mean, unit-variance, independent Gaussian distribution. The joint density is given by

$$f(x_1, x_2) = \frac{1}{2\pi} \exp[-(x_1^2 + x_2^2)/2)], \quad -\infty < x_1, \ x_2 < \infty. \quad (7)$$

The training set X consisted of 4,000 length two vectors as shown in Figure 2(a). The goal is to design a size $R = 16$ codebook to make the MSE (mean-squared error) as small as possible.

To set the design parameters we estimate D by $D = 5.2/(2 \times \log_2(16)) = 0.65$ since most of the data vectors are within a circle region of diameter 5.2. We used the weight $W^1 = 1.4$ and $\triangle W = 0.005$.

The simulation showed that the fist "hole" was selected and cut in the center of the Gaussian distribution; the 2nd

to the 4th holes were selected around the first one as shown in Figure 2(b). The algorithm continued until all 16 code vectors had been located and 16 holes had been cut. Figure 2(c) shows these code vectors and the residual data. Then the code vectors were updated by the nearest vectors of X. The "+" signs in the Figure 2(d) are the final centroids generated by the one-pass algorithm.

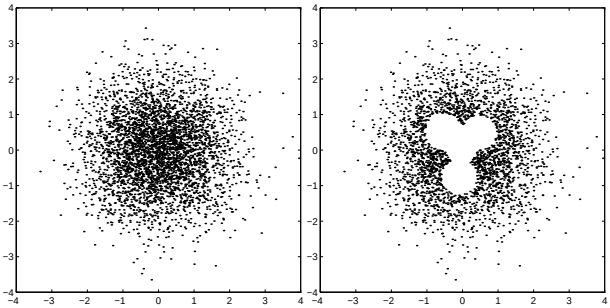

Figure 2. (a) Uncorrelated Gaussian source training data. (b) The residual data after four code vectors have been located.

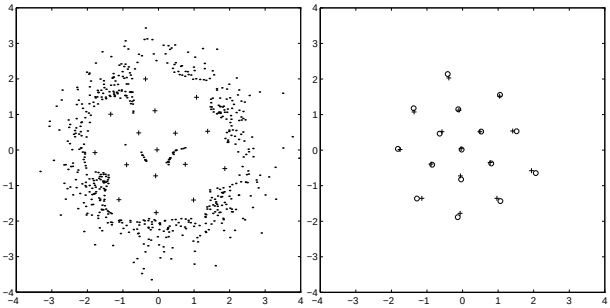

Figure 2. (c) The residual data after all 16 code vectors have been located. (d) The "+" and "o" are the centroids after one and three iterations of the LBG algorithm, respectively.

Table 1: Quantizer MSE Performance

1	One-pass VQ (proposed method)	0.218
3	One-pass VQ + 2 LBG (proposed)	0.211
2	Linde-Buzo-Gray (LBG, 20 iterations)	0.211
4	Strictly polar quantizer (from [14])	0.240
5	Unrestricted polar quantizer (from [14])	0.218
6	Scalar (Max) quantizer (from [8])	0.236
7	Dirichlet polar VQ (from [10])	0.239
8	Dirichlet rotated polar VQ (from [10])	0.222

Table 2: Comparison of One-Pass and LBG Algorithms

Type	Iterations	MFLOPS	CPU Time (seconds)	MSE
One-Pass	1	1.5	35	0.218
One-Pass +2LBG	1 + 2	3.0	67	0.211
LBG	9	7.2	166	0.218
LBG	20	15.1	333	0.211

In order to further compare the one-pass algorithm with the LBG algorithm we used the centroids designed by the one-pass algorithm as the initial centroids for the LBG algorithm, then ran two iterations of LBG (called "one-pass+2LBG"). The centroids designed by the one-pass+ 2LBG are shown in Figure 2(d) denoted by the "o"s. They are very close to the one-pass centroids ("+"s). This suggests another application of the one-pass algorithm: it can be used to determine the initial centroids for the LBG algorithm for a fast and high-performance design.

The MSE of the one-pass design is compared in Table 1 with the MSE from other methods on an uncorrelated Gaussian source with 16 centroids. The MSE of the one-pass algorithm is equal to or better than the benchmark results. Table 2 shows the CPU time and Mflops used by the one-pass and LBG algorithms. For the same MSE, 0.218, the one-pass has a speed-up-in-flops of $7.2/1.5 = 4.8$.

3.1.2. Multidimensional Gaussian Source

As shown in Table 3, the one-pass algorithm was compared with five other algorithms on an $N = 4$ iid Gaussian source of 1,600 training vectors and codebook size of $R = 16$. The speed-up-in-flops in Table 3, items 1 and 2, are from our simulations. The speed-up-in-time in items 3 to 7 were calculated based on the training time provided by Huang and Harris in [15]. Again, the one-pass algorithm shows higher speed-up.

Table 3: Comparison of Different VQ Design Approaches

	VQ Design Approaches	MSE per dimension	Speed-up
1	One-pass VQ (proposed)	0.35054	2.12
2	LBG with 5 iterations	0.34919	1.00
3	LBG (from [15])	0.41791	1.00
4	Directed-search binary-splitting [15]	0.42202	1.50
5	Pairwise nearest neighbor[15]	0.42975	2.00
6	Simulated annealing [15]	0.41166	0.0023
7	Fuzzy c-mean clustering [15]	0.51628	0.22

The speed-ups in item 1 and 2 are in-flops. All others are in-time.

3.2. Correlated Gaussian Source

The training set is 4,000 dimension two Gaussian distributed vectors with zero means, unit variances, and correlation coefficient 0.8. The codebook size is $R = 16$. The results are compared with a benchmark result in Table 4. The one-pass algorithm yields lower MSE.

Table 4: Comparison for the Correlated Gaussian Source

Types	Iterations	Mflops	CPU (seconds)	MSE
One-Pass	1	1.51	58	0.1351
One-Pass + LBG	1 + 2	2.95	110	0.1279
Block VQ (from [12])				0.1478

3.3. Laplace Source

In this example, the 4,000 dimension two training vectors have an independent Laplace distribution

$$f(x_1, x_2) = \frac{1}{2} e^{-\sqrt{2}|x_1|} e^{-\sqrt{2}|x_2|} \quad (8)$$

The training data is shown in Figure 3(a); the one-pass design's centroids ("+"s) as well as the residual data are shown in Figure 3(b). The improved centroids by one-pass+2LBG are denoted as "o"s in Figure 3(b) also.

Table 5 contains a comparison of the one-pass algorithm with several other algorithms on independent Laplace sources. The one-pass algorithm has a speed-up-in-flops of $5.8/1.5 = 3.9$ and lower MSE than the benchmark results.

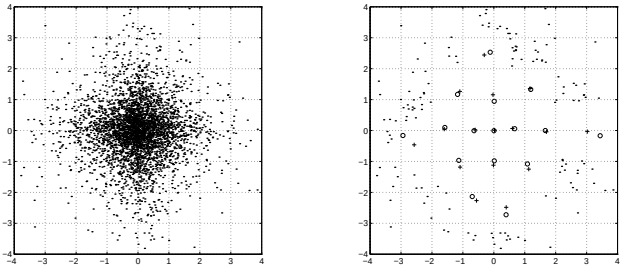

Figure 3. (a) The Laplace source training data. (b) The residual data, one-pass designed centroids "+", and one-pass+2LBG centroids "o".

Table 5: Comparison on the Laplace Source

Types	Iterations.	Mflops	CPU (sec.)	MSE
One-Pass	1	1.5	60	0.262
One-Pass + LBG	1 + 2	3.0	111	0.259
LBG (this paper)	7	5.8	208	0.260
UDQ (from [13])				0.302
MAX (from [8])				0.352
LBG (from [13])				0.264

4. CONCLUSIONS

The experimental results in this paper for different data sources showed that the one-pass algorithm results in near-optimal MSE while the CPU time and Mflops are only slightly more than that of one single iteration of the LBG algorithm.

5. ACKNOWLEDGMENTS

The authors wish to thank Dr. D.W. Tufts for many helpful discussions.

6. REFERENCES

[1] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Comm.*, vol. COM-28, pp. 84–95, Jan. 1980.

[2] Q. Li and D. W. Tufts, "Synthesizing neural networks by sequential addition of hidden nodes," in *Proc. IEEE Int'l. Conf. Neural Networks*, June 1994, pp. 708–713.

[3] Q. Li, D.W. Tufts, R.J. Duhaime, and P.V. August, "Fast training algorithms for large data sets with application to classification of multispectral images," *Proc. IEEE 28th Asilomar Conference*, CA, October 1994.

[4] Q. Li and D. W. Tufts, "Improving discriminant neural network (DNN) design by the use of principal component analysis", in *Proc. IEEE ICASSP*, May 1995, pp. 3375–3379.

[5] D. W. Tufts and Q. Li, "Principal feature classification", in Neural networks for signal processing V, *Proc. the 1995 IEEE workwhop*, August 1995.

[6] T. R. Fischer and R. M. Dicharry, "Vector quantizer design for Gaussian, gamma, and Laplacian sources," *IEEE Trans. Comm.*, vol. COM-32, pp. 1065–1069, Sept. 1984.

[7] R. M. Gray and Y. Linde, "Vector quantizers and predictive quantizers for Gauss-Markov sources," *IEEE Trans. Comm.*, vol. COM-30, pp. 381–389, Feb. 1982.

[8] J. Max, "Quantizing for minimum distortion," *IEEE Trans. Inform. Theory*, vol. IT-6, pp. 7–12, Mar. 1960.

[9] M. D. Paez and T. H. Glisson, "Minimum mean-square-error quantization in speech PCM and DPCM systems," *IEEE Trans. Comm.*, vol. COM-20, pp. 225–230, Apr. 1972.

[10] P. F. Swaszek and J. B. Thomas, "Optimal circularly symmetric quantizers," *Jour. Franklin Inst.*, vol. 313, pp. 373–384, June 1982.

[11] P. F. Swaszek, "Vector quantization for image compression," in *Proc. Princeton CISS*, Mar. 1986, pp. 254–259.

[12] P. F. Swaszek and A. Narasimhan, "Quantization of the correlated Gaussian source," in *Proc. Princeton CISS*, Mar. 1988, pp. 784–789.

[13] P. F. Swaszek, "Low dimension / moderate bitrate vector quantizers for the Laplace source," in *Abstrts. IEEE Int'l. Symp. Inform. Theory*, 1990, p. 74.

[14] S. G. Wilson, "Magnitude/phase quantization of independent Gaussian variates," *IEEE Trans. Comm.*, vol. COM-28, pp. 1924–1929, Nov. 1980.

[15] C. M. Huang and R. W. Harris, "A comparison of several vector quantization codebook generation approaches," *IEEE Trans. Image Processing,* vol. 2, pp. 108–112, Jan. 1993.