IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 6, Issue 3, Ver. III (May. - Jun. 2016), PP 48-59
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

Overview of Different Variations of Vector Quantization Schemes and Algorithms

D. Nagajyothi[1], Dr. P. Siddaiah[2]
[1]*Assoc.Prof of ECE Dept, Vardhaman College of Engineering.*
[2]*Principal, ANU College of Engg.&Tech.*

Abstract: *Exploitation of redundancy of message will be exhausted in two ways, in which one by considering the samples of message separately and second, considering the samples of message as blocks. The second way is found to be a lot of effective for both lossy and lossless compression applications. Vector quantization is a method of coding the message by forming blocks consistently. The vector quantization is being used to code speech, image and video multimedia data. The aim of this paper is to present the concept of vector quantization, significance of vector quantization as compared to that of scalar quantization and different variations of vector quantization algorithms. Different variations of vector quantization schemes are presented along with their capability and limitations.*

Keywords: *Vector quantization*

I. Introduction

Most of the compression algorithms consider the source outputs samples as an atomic entities and tries to exploit the redundancy among consequent output samples. On the other hand when the source output samples are grouped together and encoded as a block an efficient lossy and lossless compression can be obtained. It works well even for random source output samples. This kind of treating the source output samples is more advantageous particularly in lossy compression.

Here "advantageous" refers a lesser distortion for a given rate, or a lower rate for a given distortion. "Rate" means the average number of bits per input sample, and the measures of distortion will generally be the mean squared error and the signal-to-noise ratio. The idea that encoding sequences of outputs can provide an advantage over the encoding of individual samples was first put forward by Shannon, and the basic results in information theory were all proved by taking longer and longer sequences of inputs. This indicates that a quantization strategy that works with sequences or blocks of output would provide some improvement in performance over scalar quantization.

In vector quantization the source output will be grouped into blocks or vectors. For example, one can treat L consecutive samples of speech as the components of an L-dimensional vector. Or, take a block of L pixels from an image and treat each pixel value as a component of a vector of size or dimension L. This vector of source outputs forms the input to the vector quantizer. At both the encoder and decoder of the vector quantizer, a set of L-dimensional vectors are present called the *codebook* of the vector quantizer. The vectors in this codebook, known as *code-vectors*, are selected to be representative of the vectors generated from the source output. Each code-vector is assigned a binary index. At the encoder, the input vector is compared to each code-vector in order to find the code-vector closest to the input vector. The elements of this code-vector are the quantized values of the source output. In order to inform the decoder about which code-vector was found to be the closest to the input vector, the binary index of the code-vector will be transmitted or stored. Because the decoder has exactly the same codebook, it can retrieve the code-vector given its binary index. A pictorial representation of this process is shown in figure 1.

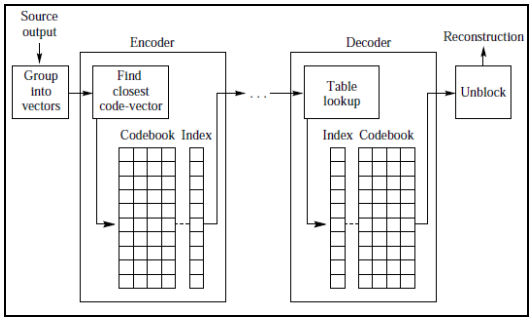

Fig. 1 Vector Quantization

Although the encoder may have to perform a considerable amount of computations in order to find the closest reproduction vector to the vector of source outputs, the decoding consists of a table lookup. This makes vector quantization a very attractive encoding scheme for applications in which the resources available for decoding are considerably less than the resources available for encoding. However, if the decoding is to be done in software, the amount of computational resources available to the decoder may be quite limited. The amount of compression will be described in terms of the rate, which will be measured in bits per sample. Suppose the size of a codebook be K,and the input vector is of dimension L.

In order to inform the decoder of which code-vector was selected, $\log_2$ K bits need to be used. For example, if the codebook contained 256code-vectors, 8 bits are needed to specify which of the 256 code-vectors had been selected at the encoder. Thus, the number of bits *per vector* is $\log_2$ Kbits. As each code-vector contains the reconstruction values for L source output samples, the number of bits *per sample* would be [$\log_2$ K]/L. Thus, the rate for an L-dimensional vector quantizer with a code book of size K is [$\log_2$ K]/L. As the measure of distortion that will be used is the mean squared error. In a codebook **C**, containing the K code-vectors {Yi}, the input vector X is closest to Yj,

$\| X - Y_j \|^2 \leq \| X - Y_i \|^2$ for all $Y_i \in \mathbf{C}$

Where $X = (x_1 x_2 \ldots x_L)$ and

$$\| X \|^2 = \sum_{i=1}^{L} x_i^2$$

The term *sample* will always refer to a scalar value. The output points of the quantizer are often referred to as *levels*. Thus, a quantizer will be referred with K output points or code-vectors, or as a K-level quantizer.

II. Advantage Of Vector Quantization Over Scalar Quantization:

For a given rate (in bits per sample), use of vector quantization results in a lower distortion than when scalar quantization is used at the same rate, for several reasons [1][2][3]. If the source output is correlated, vectors of source output values will tend to fall in clusters. By selecting the quantizer output points to lie in these clusters, a more accurate representation of the source output is possible. Consider the following example. Suppose the height of individuals varied uniformly between 40 and 80 inches, and the weight varied uniformly between 40 and 240 pounds.

Assume that a total of 6 bits are allowed to represent each pair of values. One use 3 bits to quantize the height and 3 bits to quantize the weight. Thus, the weight range between 40 and 240 pounds would be divided into eight intervals of equal width of 25 and with reconstruction values {52,77 . . . 227}.Similarly, the height range between 40 and 80 inches can be divided into eight intervals of width five, with reconstruction levels {42, 47, . . ., 77}.

This approach seems reasonable when one look at the representation of height and weight separately. But let's look at this quantization scheme in two dimensions, and plot the height values along the x-axis and the weight values along the y-axis. The height values are still being quantized to the same eight different values, as are the weight values. The two-dimensional representation of these two quantizers is shown in figure 2.

From the figure it is evident that one can effectively get a quantizer output for a person who is 80 inches (6 feet 8 inches) tall and weighs 40 pounds, as well as a quantizer output for an individual whose height is 42 inches but weighs more than 200 pounds. Obviously, these outputs will never be used, as is the case for many of the other outputs. A more sensible approach would be to use a quantizer like the one shown in figure 3, where the fact that the height and weight are correlated is accounted for.

This quantizer has exactly the same number of output points as the quantizer in figure 2; however, the output points are clustered in the area occupied by the input. Using this quantizer, one can no longer quantize the height and weight separately. Considering them as the coordinates of a point in two dimensions in order to find the closest quantizer output point is a good option. However, this method provides a much finer quantization of the input.

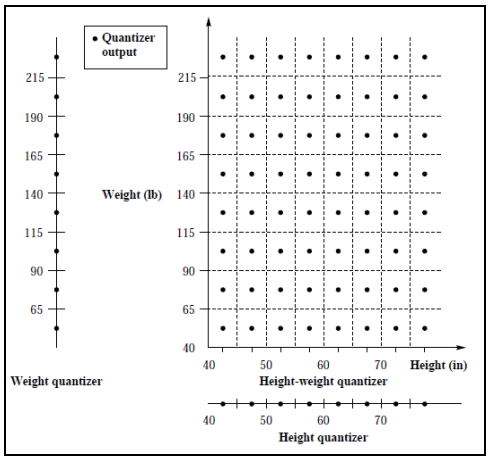

Fig. 2 The two-dimensional representation of two quantizers

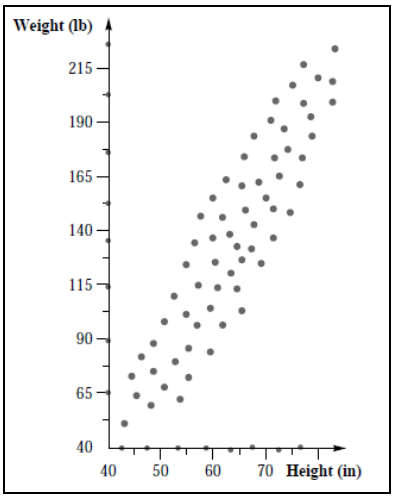

Fig. 3 The height-weight vector quantizer

III. Concept Of Vector Quantization

In the example considered in introduction section, the way of exploiting the structure in the source output is to place the quantizer output points where the source output are most likely to congregate. The set of quantizer output points is called the *codebook* of the quantizer, and the process of placing these output points is often referred to as *codebook design*. When the source output was grouped in two-dimensional vectors, as in the case of above example, one might be able to obtain a good codebook design by plotting a representative set of source output points and then visually locate where the quantizer output points should be. However, this approach to codebook design breaks down when higher-dimensional vector quantizers are designed. Consider designing the codebook for a 16-dimensional quantizer. Obviously, a visual placement approach will not work in this case. An automatic procedure for locating where the source outputs are clustered is needed. The most popular approach to designing vector quantizers is a clustering procedure known as the k-means algorithm, which was developed for pattern recognition applications.

The k-means algorithm functions as follows: Given a large set of output vectors from the source, known as the *training set*, and an initial set of k representative patterns, assign each element of the training set to the closest representative pattern. After an element is assigned, the representative pattern is updated by computing the centroid of the training set vectors assigned to it. When the assignment process is complete, we will have k groups of vectors clustered around each of the output points. Stuart Lloyd [4] used this approach to generate the *pdf*-optimized scalar quantizer, except that instead of using a training set, he assumed that the distribution was known. The Lloyd algorithm functions as follows:

1. Start with an initial set of reconstruction values $\{y_i^{(0)}\}_{i=1}^{M}$. Set k = 0, $D^{(0)}$= 0. Select threshold $\in$.

2. Find decision boundaries

$$b_j^{(k)} = \frac{y_{j+1}^{(k)} + y_j^{(k)}}{2}, \text{ j = 1,2. . . M-1.}$$

3. Compute the distortion

$$D^{(k)} = \sum_{i=1}^{M} \int_{b_{i-1}^{(k)}}^{b_i^{(k)}} (x - y_i)^2 \, f_X(x)\, dx.$$

4. If $D^{(k)} - D^{(k-1)} < \in$, stop; otherwise, continue.

5. k= k+1. Compute new reconstruction values

$$y_j^{(k)} = \frac{\int_{b_{j-1}^{(k-1)}}^{b_j^{(k-1)}} x\, f_X(x)\, dx}{\int_{b_{j-1}^{(k-1)}}^{b_j^{(k-1)}} f_X(x)\, dx}$$

Go to Step 2.

Linde, Buzo, and Gray generalized this algorithm to the case where the inputs are no longer scalars [5]. For the case where the distribution is known, the algorithm looks very much like the Lloyd algorithm described above.

1. Start with an initial set of reconstruction values $\{Y_i^{(0)}\}_{i=1}^{M}$. Set k = 0, $D^{(0)}= 0$. Select threshold $\in$.

2. Find quantization regions

$$V_i^{(k)} = \{X : d(X, Y_i) < d(X, Y_j) \ \forall \ j \neq i\} \qquad \text{j = 1, 2. . . M.}$$

3. Compute the distortion

$$D^{(k)} = \sum_{i=1}^{M} \int_{V_i^{(k)}} \left\| X - Y_i^{(k)} \right\|^2 f_X(X)\, dX.$$

4. If $\frac{D^{(k)} - D^{(k-1)}}{D^{(k)}} < \in$, stop; otherwise, continue.

5. k= k+1. Find new reconstruction values $\{Y_i^{(k)}\}_{i=1}^{M}$ that are the centroids of $\{V_i^{(k-1)}\}$. Go to Step 2.

This algorithm is not very practical because the integrals required to compute the distortions and centroids are over odd-shaped regions in n dimensions, where n is the dimension of the input vectors. Generally, these integrals are extremely difficult to compute, making this particular algorithm more of an academic interest. Of more practical interest is the algorithm for the case where we have a training set available. In this case, the algorithm looks very much like the k-means algorithm.

1. Start with an initial set of reconstruction values $\{Y_i^{(0)}\}_{i=1}^{M}$ and a set of training vectors $\{X_n\}_{n=1}^{N}$. Set k = 0, $D^{(0)}= 0$. Select threshold $\in$.

2. The quantization regions $\{V_i^{(k)}\}_{i=1}^{M}$ are given by

$$V_i^{(k)} = \{X_n : d(X_n, Y_i) < d(X_n, Y_j) \ \forall \ j \neq i\} \quad \text{i = 1, 2. . . M.}$$

Assume that none of the quantization regions are empty.

3. Compute the average distortion $D^{(k)}$ between the training vectors and the representative reconstruction value.

4. If $\frac{D^{(k)} - D^{(k-1)}}{D^{(k)}} < \in$, stop; otherwise, continue.

5. k= k+1. Find new reconstruction values $\{Y_i^{(k)}\}_{i=1}^{M}$ that are average value of the elements of each of the quantization regions $\{V_i^{(k-1)}\}$. .Go to Step 2.

This algorithm forms the basis of most vector quantizer designs. It is popularly known as the Linde-Buzo-Gray or LBG algorithm, or the generalized Lloyd algorithm (GLA) [5].Although the paper of Linde, Buzo, and Gray [5] is a starting point for most of the work on vector quantization, the latter algorithm had been used several years prior by Edward E. Hilbert at the NASA Jet Propulsion Laboratories in Pasadena, California. Hilbert's starting point was the idea of clustering, and although he arrived at the same algorithm as described above, he called it the *cluster compression algorithm* [6].In order to see how this algorithm functions, consider

the following example of a two dimensional vector quantizer codebook design. Suppose the training set consists of the height and weight values shown in table 1.

Table 1: Training set for designing vector quantizer codebook

Height	Weight
72	180
65	120
59	119
64	150
65	162
57	88
72	175
44	41
62	114
60	110
56	91
70	172

The initial set of output points is shown in table 2. The inputs, outputs, and quantization regions are shown in figure 4.

Table 2: Initial set of output points for codebook design

Height	Weight
45	50
75	117
45	117
80	180

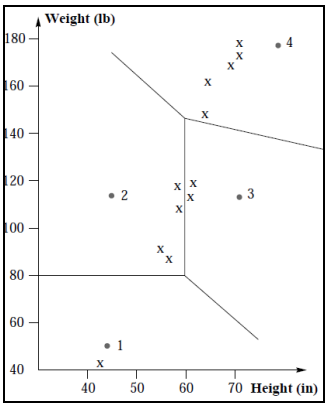

Fig. 4 Initial state of the quantizer

The input (44, 41) has been assigned to the first output point; the inputs (56, 91), (57,88), (59, 119), and (60, 110) have been assigned to the second output point; the inputs(62, 114), and (65, 120) have been assigned to the third output; and the five remaining vectors from the training set have been assigned to the fourth output. The distortion for this assignment is 387.25. There is only one vector in the first quantization region, so the first output point is (44, 41). The average of the four vectors in the second quantization region (rounded up) is the vector (58, 102), which is the new second output point. In a similar manner, the third and fourth output points can be computed as (64, 117) and (69, 168). The new output points and the corresponding quantization regions are shown in figure 5.

From figure 5, it is evident that while the training vectors that were initially part of the first and fourth quantization regions are still in the same quantization regions, the training vectors (59,115) and (60,120), which were in quantization region 2, are now in quantization region 3. The distortion corresponding to this assignment of training vectors to quantization regions is 89, considerably less than the original 387.25.

The first and fourth output points do not change because the training vectors in the corresponding regions have not changed. However, the training vectors in regions 2 and 3 have changed. Re-computing the output points for these regions, result in (57, 90) and (62, 116). The final form of the quantizer is shown in figure 6. The distortion corresponding to the final assignments is 60.17.

The LBG algorithm is conceptually simple, and the resulting vector quantizer is remarkably effective in the compression of a wide variety of inputs, both by itself and in conjunction with other schemes. One application for which the vector quantizer is extremely popular is image compression. For image compression,

the vector is formed by taking blocks of pixels of size N ×M and treating them as an L = NM dimensional vector. Consider N =M. Instead of forming vectors in this manner, vectors can be formed by taking L pixels in a row of the image. However, this does not take advantage of the two dimensional correlations in the image. Recall that correlation between the samples provides the clustering of the input, and the LBG algorithm takes advantage of this clustering.

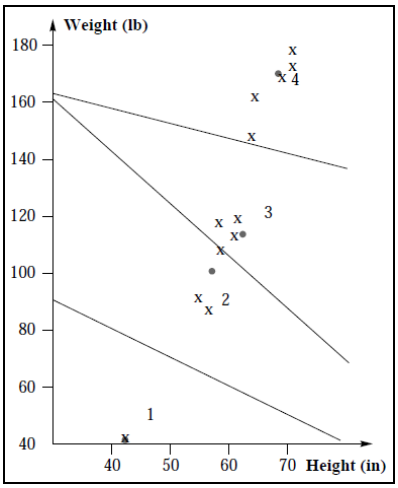

Fig. 5 Quantizer after first iteration

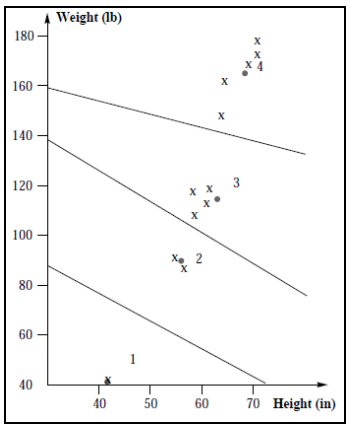

Fig. 6 Final state of the quantizer

If each codeword in the codebook is a vector with L elements and if B bits are used to represent each element, then in order to transmit the codebook of a K-level quantizer B×L×K bits are needed. Consider B = 8 and L = 16. Therefore, K×128 bits are needed to transmit the codebook. As image consists of256×256 pixels, the overhead in bits per pixel is 128K/65,536. The overhead for different values of K is summarized in table 3.

Table 3: Overhead in bits/pixel for codebooks of different sizes

Codebook size K	Overhead in bits/pixel
16	0.03125
64	0.125
256	0.5
1024	2.0

While the overhead for a codebook of size 16 seems reasonable, the overhead for a codebook of size 1024 is over three times the rate required for quantization. Given the excessive amount of overhead required for sending the codebook along with the vector quantized image, there has been substantial interest in the design of codebooks that are more generic in nature and, therefore, can be used to quantize a number of images. As expected, the reconstructed images are not of the same quality as when the codebook is generated from the image to be quantized. However, this is only true as long as the overhead required for storage or transmission of the codebook is ignored.

Assume the rate of encoding a source is R bits per sample; that is, the average number of bits per sample in the compressed source output is R. If an L-dimensional quantizer is to be used L samples will be

grouped together into vectors. This means that we would have RL bits available to represent each vector. With RL bits, 2^{RL} different output vectors can be represented.

In many applications, such as low-rate speech coding, where low rate operations are involved this is not a drawback. However, for applications such as high-quality video coding, which requires higher rates, this is definitely a problem. There are several approaches to solving these problems. Each entails the introduction of some structure in the codebook and/or the quantization process. While the introduction of structure mitigates some of the storage and computational problems, there is generally a trade-off in terms of the distortion performance. Some of these approaches are presented in the following section.

IV. Variants Of Vector Quantization:

a. Tree-structured Vector Quantizers

Consider the two-dimensional vector quantizer where the output points in each quadrant are the mirror image of the output points in neighboring quadrants. Given an input to this vector quantizer, the number of comparisons necessary for finding the closest output point can be reduced by using the sign on the components of the input. The sign on the components of the input vector will tell in which quadrant the input lies. Because all the quadrants are mirror images of the neighboring quadrants, the closest output point to a given input will lie in the same quadrant as the input itself.

Therefore, we only need to compare the input to the output points that lie in the same quadrant, thus reducing the number of required comparisons by a factor of four. This approach can be extended to L dimensions, where the signs on the L components of the input vector can tell in which of the 2^L hyper-quadrants the input lies, which in turn would reduce the number of comparisons by 2^L. This approach works well when the output points are distributed in a symmetrical manner. However, it breaks down as the distribution of the output points becomes less symmetrical.

b. Structured Vector Quantizers

The tree-structured vector quantizer solves the complexity problem, but acerbates the storage problem. The correlation between samples of the output of a source leads to clustering. This clustering is exploited by the LBG algorithm by placing output points at the location of these clusters. However, even when there is no correlation between samples, there will be a kind of probabilistic structure that becomes more evident as the random inputs of a source are grouped into larger and larger blocks or vectors.

For spherically symmetrical distributions like the Gaussian distribution, the contours of constant probability are circles in two dimensions, spheres in three dimensions, and hyper spheres in higher dimensions. The points away from the origin have very little probability mass associated with them. The points on constant probability contours farther away from the origin have very little probability mass associated with them. Therefore, one can get rid of all of the points outside some contour of constant probability without incurring much of a distortion penalty. In addition as the number of reconstruction points is reduced, there is a decrease in rate, thus improving the rate distortion performance.

i. Pyramid Vector Quantization

As the dimension of the input vector increases, something interesting happens. Consider quantization of a random variable X with *pdf* $f_x(x)$ and differential entropy h(X). Suppose we block samples of this random variable into a vector **X**. A result of Shannon's, called the *asymptotic equipartition property* (AEP), states that for sufficiently large L and arbitrarily small $\in$.

$$\left|\frac{\log f_X(X)}{L} + h(X)\right| < \in$$

for all but a set of vectors with a vanishingly small probability [7]. This means that almost all the L-dimensional vectors will lie on a contour of constant probability given by

$$\left|\frac{\log f_X(X)}{L}\right| = -h(X)$$

Given that this is the case, Sakrison [8] suggested that an optimum manner to encode the source would be to distribute 2^{RL} points uniformly in this region. Fischer [9] used this insight to design a vector quantizer called the *pyramid vector quantizer* for the Laplacian source. The vector quantizer consists of points of the rectangular quantizer that fall on the hyper pyramid given by

$$\sum_{i=1}^{L} | x_i | = C$$

where C is a constant depending on the variance of the input. Shannon's result is a symptotic, and for realistic values of L, the input vector is generally not localized to a single hyper pyramid. For this case, Fischer first finds the distance

$$r = \sum_{i=1}^{L} | x_i | .$$

This value is quantized and transmitted to the receiver. The input is normalized by this gain term and quantized using a single hyper pyramid. The quantization process for the shape term consists of two stages: finding the output point on the hyper pyramid closest to the scaled input, and finding a binary codeword for this output point [9]. This approach is quite successful, and for a rate of 3 bits per sample and a vector dimension of 16, one can obtain an SNR value of 16.32dB. If the vector dimension was increased to 64, SNR value of 17.03dB can be obtained. Compared to the SNR obtained from using a non-uniform scalar quantizer, this is an improvement of more than 4dB.Note that in this approach the input vector was separated into a *gain* term and a pattern or *shape* term. Quantizers of this form are called *gain-shape vector quantizers*, or *productcode vector quantizers* [10].

ii. **Polar and Spherical Vector Quantizers**

For the Gaussian distribution, the contours of constant probability are circles in two dimensions and spheres and hyper spheres in three and higher dimensions. In two dimensions, the input vector can be quantized by first transforming it into polar coordinates r and θ:

$$r = \sqrt{x_1^2 + x_2^2} \text{ and } \theta = \tan^{-1} \frac{x_2}{x_1}.$$

r and θ can then be either quantized independently [11], or the quantized value of r can be used as an index to a quantizer for θ [12]. The former is known as a polar quantizer; the latter, an unrestricted polar quantizer. The advantage to quantizing r and θ independently isone of simplicity. The quantizes for r and θ are independent scalar quantizers. However, the performance of the polar quantizers is not significantly higher than that of scalar quantization of the components of the two-dimensional vector. The unrestricted polar quantizer has amore complex implementation, as the quantization of θ depends on the quantization of r. However, the performance is also somewhat better than the polar quantizer. The polar quantizer can be extended to three or more dimensions [13].

iii. **Lattice Vector Quantizers**

Recall that quantization error is composed of two kinds of error, overload error and granular error. The overload error is determined by the location of the quantization regions furthest from the origin, or the boundary. In scalar quantization, the granular error was determined by the size of the quantization interval. In vector quantization, the granular error is affected by the size and shape of the quantization interval. Consider the square and circular quantization regions shown in figure 7.

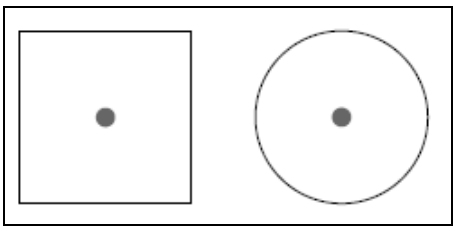

Fig. 7 Square and circular quantization regions

These quantization regions need to be distributed in a regular manner over the space of source outputs. However, for now, consider the quantization region at the origin. Let's assume they both have the same area so that they can be compared with each other. This way it would require the same number of quantization regions to cover a given area. That is, two quantization regions of the same size will be compared. To have an area of one, the square has to have sides of length one. As the area of a circle is given by πr^2, the radius of the circle is $\frac{1}{\sqrt{\pi}}$. The maximum quantization error possible with the square quantization region is when the input is at one of the four corners of the square. In this case, the error is $1/\sqrt{2}$ or about 0.707. For the circular

quantization region, the maximum error occurs when the input falls on the boundary of the circle. In this case, the error is $\frac{1}{\sqrt{\pi}}$ or about 0.56. Thus, the maximum granular error is larger for the square region than the circular region. In general, one is more concerned with the average squared error than the maximum error. Computing the average squared error:

$$\int_{Square} \| X \|^2 \, dX = 0.166\overline{6}.$$

$$\int_{Circle} \| X \|^2 \, dX = 0.159.$$

Thus, the circular region would introduce less granular error than the square region. Unfortunately, a basic requirement for the quantizer is that for every possible input vector there should be a unique output vector. In order to satisfy this requirement and have a quantizer with sufficient structure that can be used to reduce the storage space, a union of translates of the quantization region should cover the output space of the source. In other words, the quantization region should *tile* space. A two-dimensional region can be tiled by squares, but it cannot be tiled by circles. Apart from squares, other shapes that tile space include rectangles and hexagons. It turns out that the best shape to pick for a quantization region in two dimensions is a hexagon [14].In two dimensions, it is relatively easy to find the shapes that tile space, then select the one that gives the smallest amount of granular error.

However, when we start looking at higher dimensions, it is difficult, if not impossible, to visualize different shapes, let alone find which ones tile space. An easy way out of this dilemma is to remember that a quantizer can be completely defined by its output points. In order for this quantizer to possess structure, these points should be spaced in some regular manner. Regular arrangements of output points in space are called *lattices*. Mathematically, a lattice can be defined as follows: Let $\{a_1, a_2, ..., a_L\}$ be L independent L-dimensional vectors. Then the set

$$L = \left\{ x : x = \sum_{i=1}^{L} u_i a_i \right\}$$

is a lattice if $\{u_i\}$ are all integers. When a subset of lattice points is used as the output points of a vector quantizer, the quantizer is known as a *lattice vector quantizer*. From this definition, the pyramid vector quantizer described earlier can be viewed as a lattice vector quantizer. Basing a quantizer on a lattice solves the storage problem. As any lattice point can be regenerated if the basis set is known, there is no need to store the output points. Further, the highly structured nature of lattices makes finding the closest output point to an input relatively simple.

c. Gain-Shape Vector Quantization

In some applications such as speech, the dynamic range of the input is quite large. One effect of this is that, in order to be able to represent the various vectors from the source, a very large codebook is needed. This requirement can be reduced by normalizing the source output vectors, then quantizing the normalized vector and the normalization factor separately [15][10]. In this way, the variation due to the dynamic range is represented by the normalization factor or *gain*, while the vector quantizer is free to do what it does best, which is to capture the structure in the source output. Vector quantizers that function in this manner are called *gain-shape vector quantizers*. The pyramid quantizer discussed earlier is an example of a gain-shape vector quantizer.

d. Mean-Removed Vector Quantization

If a codebook is being generated from an image, differing amounts of background illumination would result in vastly different codebooks. This effect can be significantly reduced if the mean from each vector is removed before quantization. The mean and the mean-removed vector can then be quantized separately. The mean can be quantized using a scalar quantization scheme, while the mean-removed vector can be quantized using a vector quantizer. If this strategy is used, the vector quantizer should be designed using mean-removed vectors as well.

e. Classified Vector Quantization

Sometimes the source output can be divided into separate classes with different spatial properties. In these cases, it can be very beneficial to design separate vector quantizers for the different classes. This approach, referred to as *classified vector quantization*, is especially useful in image compression, where edge and non-edge regions form two distinct classes. The training set can be separated into vectors that contain edges and vectors that do not. A separate vector quantizer can be developed for each class. During the encoding process,

the vector is first tested to see if it contains an edge. A simple way to do this is to check the variance of the pixels in the vector. A large variance will indicate the presence of an edge. More sophisticated techniques for edge detection can also be used. Once the vector is classified, the corresponding codebook can be used to quantize the vector. The encoder transmits both the label for the codebook used and the label for the vector in the codebook[16].A slight variation of this strategy is to use different kinds of quantizers for the different classes of vectors. For example, if certain classes of source outputs require quantization at a higher rate than is possible using LBG vector quantizers, lattice vector quantizers can be used [17].

f. Multistage Vector Quantization

Multistage vector quantization [18] is an approach that reduces both the encoding complexity and the memory requirements for vector quantization, especially at high rates. In this approach, the input is quantized in several stages. In the first stage, a low-rate vector quantizer is used to generate a coarse approximation of the input. This coarse approximation, in the form of the label of the output point of the vector quantizer, is transmitted to the receiver. The error between the original input and the coarse representation is quantized by the second-stage quantizer, and the label of the output point is transmitted to the receiver. In this manner, the input to the nth-stage vector quantizer is the difference between the original input and the reconstruction obtained from the outputs of the preceding n−1 stages. The difference between the input to a quantizer and the reconstruction value is often called the *residual*, and the multistage vector quantizers are also known as *residual vector quantizers*[19]. The reconstructed vector is the sum of the output points of each of the stages. Suppose we have a three-stage vector quantizer, with the three quantizers represented by $\mathbf{Q_1}$,$\mathbf{Q_2}$, and$\mathbf{Q_3}$. Then for a given input **X**,

$$Y_1 = Q_1(X)$$

$$Y_2 = Q_2(X - Q_1(X))$$

$$Y_3 = Q_3(X - Q_1(X) - Q_2(X - Q_1(X)))$$

The reconstruction $\hat{X}$ is given by

$$\hat{X} = Y_1 + Y_2 + Y_3$$

This process is shown in figure 8.

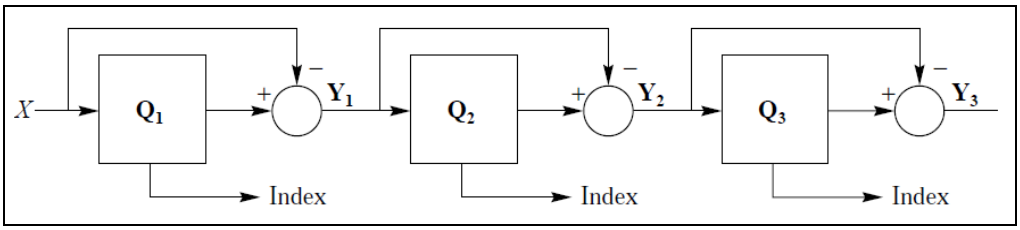

Fig. 8 Multistage vector quantization

If there are K stages, and the codebook size of the n^{th}-stage vector quantizer is L_n, then the effective size of the overall codebook is $L_1 \times L_2 \times \cdots \times L_K$. However, only $L_1+L_2+\cdots+L_K$ vectors need to be stored, which is also the number of comparisons required. Consider a five-stage vector quantizer, each with a codebook size of 32, meaning that 160 codewords have to be stored. This would provide an effective codebook size of 32^5= 33,554,432. The computational savings are also of the same order. This approach allows to use vector quantization at much higher rates. However, at rates at which it is feasible to use LBG vector quantizers, the performance of the multistage vector quantizers is generally lower than the LBG vector quantizers [20]. The reason for this is that after the first few stages, much of the structure used by the vector quantizer has been removed, and the vector quantization advantage that depends on this structure is not available [19][21].

There may be some vector inputs that can be well represented by fewer stages than others. A multistage vector quantizer with a variable number of stages can be implemented by extending the idea of recursively indexed scalar quantization to vectors. It is not possible to do this directly because there are some fundamental differences between scalar and vector quantizers. The input to a scalar quantizer is assumed to be *iid*. On the other hand, the vector quantizer can be viewed as a pattern-matching algorithm [22]. The input is assumed to be one of a number of different patterns. The scalar quantizer is used after the redundancy has been removed from the source sequence, while the vector quantizer takes advantage of the redundancy in the data. With these differences in mind, the recursively indexed vector quantizer (RIVQ) can be described as a two-stage process. The first stage performs the normal pattern-matching function, while the second stage recursively quantizes the residual if the magnitude of the residual is greater than some pre-specified threshold. The codebook of the

second stage is ordered so that the magnitude of the codebook entries is a non-decreasing function of its index. Then an index I that will determine the mode can be chosen in which the RIVQ operates.

g. Adaptive Vector Quantization

While LBG vector quantizers function by using the structure in the source output, this reliance on the use of the structure can also be a drawback when the characteristics of the source change over time. For situations like these, we would like to have the quantizer adapt to the changes in the source output. For mean-removed and gain-shape vector quantizers, the scalar aspect of the quantizer can be adapted. One way of adapting the codebook to changing input characteristics is to start with a very large codebook designed to accommodate a wide range of source characteristics [23].This large codebook can be ordered in some manner known to both transmitter and receiver. Given a sequence of input vectors to be quantized, the encoder can select a subset of the larger codebook to be used. Information about which vectors from the large codebook were used can be transmitted as a binary string.

For example, if the large codebook contained 10vectors, and the encoder was to use the second, third, fifth, and ninth vectors, the binary string 0110100010 can be sent, with a 1 representing the position of the codeword used in the large codebook. This approach permits the use of a small codebook that is matched to the local behavior of the source. This approach can be used with particular effectiveness with the recursively indexed vector quantizer [24]. Recall that in the recursively indexed vector quantizer, the quantized output is always within a prescribed distance of the inputs, determined by the index i. This means that the set of output values of the RIVQ can be viewed as an accurate representation of the inputs and their statistics. Therefore a subset of the output set of the previous intervals can be treated as the large codebook.

Then the method described in [23] can be used to inform the receiver of which elements of the previous outputs form the codebook for the next interval. This method is quite simple. Suppose an output set, in order of first appearance, is {p, a, q, s, l, t, r}, and the desired codebook for the interval to be encoded is {a, q, l, r}. Then we would transmit the binary string 0110101 to the receiver. The 1s correspond to the letters in the output set, which would be elements of the desired codebook. The subset for the current interval will be selected by finding the closest vectors from the collection of past outputs to the input vectors of the current set. This means that there is an inherent delay of one interval imposed by this approach. The overhead required to send the codebook selection is M/N, where M is the number of vectors in the output set and Nis the interval size.

h. Some more variants of Vector quantizers

D. Martinez Munoz et al. proposed a low delay audio coder using adaptive vector quantization [25]. A new coder that responds to a sub band ADPCM structure and incorporates adaptive vector quantization was presented, requiring only 2 bits/sample. Each vector component is weighted according to its estimated standard deviation. The introduced coding delay is less than 2 milliseconds, and is only due to the perfect reconstruction filter bank. In order to obtain the best configuration, results have been obtained varying the number of bands, the number of bits in the quantizer and the analysis filters.

Davor Petrinovic et al., proposed a modification of a classical predictive vector quantization (PVQ) technique with switched-adaptive prediction for line spectrum frequencies (LSF) quantization [26]. Lower complexity is achieved through use of higher number of switched prediction matrices but with reduced number of their nonzero elements. The structures of such matrices and optimal matrix elements are obtained to maximize the quantizer closed-loop prediction gain. A comparison of this quantizer to the ones with full prediction matrices as well as to the quantizer incorporating diagonal matrices is given.

T. Z. Shabestary et al.presented an encoding scheme based on random coding theory in conjunction with companding techniques [27]. This combination provides an easy design, and a low-storage quantizer. The groundwork of the study is Gaussian sources. The motivation is the attractive properties of Gaussian and Gaussian mixtures (GMs), and their applications in data compression. Here, codebooks are generated by companding a union of randomly translated cubic lattices. The compander functions operate scalar wise, providing an affordable encoding complexity that is rate independent. For such a codebook, an optimal search, and the indexing of the code vectors are addressed. Performance close to finite-dimensional random coding is achieved, while the complexity is not far from scalar coding. The structure of the codebook balances between complexity and performance. In comparison to some other methods, the advantage of the proposed scheme is more distinct for correlated sources.

Jung Hoon Lee et al. propose a two-stage vector channel quantizer for multiple-input multiple-output (MIMO) broadcast channels with limited feedback [28]. When the number of total users is larger than the number of transmit antennas, the users to be served are selected and then the selected users are supported by beam forming based on quantized feedback information. If channel gain information (CGI) and channel direction information (CDI) are independently quantized with a product codebook, as in conventional channel quantization, the effect of CDI quantization errors is boosted by the CGI value, which becomes more pronounced as the number of users increases because the selected users are likely to have larger CGI than the

others. Motivated by the analysis, a new two-stage quantizer where CGI is quantized at the first stage was devised, and then CDI is adaptively quantized at the second stage according to the quantized CGI value and total number of users. The two-stage quantizer for an arbitrarily given CGI quantizer and the number of total users was optimized. It is demonstrated that for the same feedback size, the proposed quantizer markedly improves conventional quantization with a product codebook in terms of average sum rate in MIMO BC.

V. Conclusions

The complex part of coding using vector quantization is the design of code book. The amount of redundancy that may be exploited is directly depend upon the potential of code book. The code book has to be designed by considering many factors. The type of information, the source, rate at which to code and several things must be considered. This paper has concerned the concept of vector quantization and the variants of vector quantization are presented. The superiority of vector quantization over scalar quantization is presented. The delay, code rate and complexity are conflicting each other in the application of vector quantization for coding a multimedia data. An optimum design has to be made to have less delay, less complexity and high rate.

References

[1] T.M. Cover and J.A. Thomas. Elements of Information Theory. Wiley Series in Telecommunications. John Wiley & Sons Inc., 1991.
[2] T. Berger. Rate Distortion Theory: A Mathematical Basis for Data Compression. Prentice-Hall, Englewood Cliffs, NJ, 1971.
[3] R.M. Gray. Entropy and Information Theory. Springer-Verlag, 1990.
[4] S.P. Lloyd. Least Squares Quantization in PCM. IEEE Transactions on Information Theory, IT-28:127–135, March 1982.
[5] Y. Linde, A. Buzo, and R.M. Gray. An algorithm for vector quantization design. IEEE Transactions on Communications, COM-28:84–95, Jan. 1980.
[6] E.E. Hilbert. Cluster Compression Algorithm—A Joint Clustering Data Compression Concept. Technical Report JPL Publication 77-43, NASA, 1977.
[7] C.E. Shannon. A Mathematical Theory of Communication. Bell System Technical Journal, 27:379–423, 623–656, 1948.
[8] D.J. Sakrison. A Geometric Treatment of the Source Encoding of a Gaussian Random Variable. IEEE Transactions on Information Theory, IT-14(481–486):481–486, May 1968.
[9] T.R. Fischer. A Pyramid Vector Quantizer. IEEE Transactions on Information Theory, IT-32:568–583, July 1986.
[10] M.J. Sabin and R.M. Gray. Product Code Vector Quantizers for Waveform and Voice Coding. IEEE Transactions on Acoustics, Speech, and Signal Processing, ASSP- 32:474–488, June 1984.
[11] W.A. Pearlman. Polar Quantization of a Complex Gaussian Random Variable. IEEE Transactions on Communications, COM-27:892–899, June 1979.
[12] S.G. Wilson. Magnitude Phase Quantization of Independent Gaussian Variates. IEEE Transactions on Communications, COM-28:1924–1929, November 1980.
[13] P.F. Swaszek and J.B. Thomas. Multidimensional Spherical Coordinates Quantization. IEEE Transactions on Information Theory, IT-29:570–575, July 1983.
[14] D.J. Newman. The Hexagon Theorem. IEEE Transactions on Information Theory, IT-28:137–139, March 1982.
[15] A. Buzo, A.H. Gray, R.M. Gray, and J.D. Markel. Speech Coding Based Upon Vector Quantization. IEEE Transactions on Acoustics, Speech, and Signal Processing, ASSP-28: 562–574, October 1980.
[16] B. Ramamurthi and A. Gersho. Classified Vector Quantization of Images. IEEE Transactions on Communications, COM-34:1105–1115, November 1986.
[17] V. Ramamoorthy and K. Sayood. A Hybrid LBG/Lattice Vector Quantizer for High Quality Image Coding. In E. Arikan, editor, Proc. 1990 Bilkent International Conference on New Trends in Communication, Control and Signal Processing. Elsevier, 1990.
[18] B.H. Juang and A.H. Gray. Multiple Stage Vector Quantization for Speech Coding. In Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing, pages 597–600. IEEE, April 1982.
[19] C.F. Barnes and R.L. Frost. Residual Vector Quantizers with Jointly Optimized Code Books. In Advances in Electronics and Electron Physics, pages 1–59. Elsevier, 1992.
[20] A. Gersho and R.M. Gray. Vector Quantization and Signal Compression. Kluwer Academic Publishers, 1991.
[21] C.F. Barnes and R.L. Frost. Vector quantizers with direct sum codebooks. IEEE Transactions on Information Theory, 39:565–580, March 1993.
[22] A. Gersho and V. Cuperman. A Pattern Matching Technique for Speech Coding. IEEE Communications Magazine, pages 15–21, December 1983.
[23] S. Panchanathan and M. Goldberg. Adaptive Algorithm for Image Coding Using Vector Quantization. Signal Processing: Image Communication, 4:81–92, 1991.
[24] A.G. Al-Araj and K. Sayood. Vector Quantization of Nonstationary Sources. In Proceedings International Conference on Telecommunications—1994, pages 92–95. IEEE, 1994.
[25] D. Martinez Munoz, M. Rosa Zurera, F. Lopez Ferreras and N. Ruiz Reyes, "Low delay audio coder using adaptive vector quantization", 2000 10th European Signal Processing Conference, pp. 1-4, Sept., 2000.
[26] Davor Petrinovic and Davor Petrinovic, "Reduced complexity LSF vector quantization with switched-adaptive prediction", Proceedings of the 3rd International Symposium on Image and Signal Processing and Analysis, pp. 1039 - 1044 Vol.2, Sept. 2003.
[27] T. Z. Shabestary and P. Hedelin, "Vector quantization by companding a union of Z-lattices", IEEE Transactions on Information Theory, vol. 51, Iss. 2, pp. 738 – 746, Feb 2005.
[28] Jung Hoon Lee and Wan Choi, "Unified Codebook Design for Vector Channel Quantization in MIMO Broadcast Channels", IEEE Transactions on Signal Processing, vol. 63, Iss. 10, pp. 2509-2519, March 2015.