Exhibit F

Vector Quantization

Robert M. Gray

A vector quantizer is a system for mapping a sequence of continuous or discrete vectors into a digital sequence suitable for communication over or storage in a digital channel. The goal of such a system is data compression: to reduce the bit rate so as to minimize communication channel capacity or digital storage memory requirements while maintaining the necessary fidelity of the data. The mapping for each vector may or may not have memory in the sense of depending on past actions of the coder, just as in well established scalar techniques such as PCM, which has no memory, and predictive quantization, which does. Even though information theory implies that one can always obtain better performance by coding vectors instead of scalars, scalar quantizers have remained by far the most common data compression system because of their simplicity and good performance when the communication rate is sufficiently large. In addition, relatively few design techniques have existed for vector quantizers.

During the past few years several design algorithms have been developed for a variety of vector quantizers and the performance of these codes has been studied for speech waveforms, speech linear predictive parameter vectors, images, and several simulated random processes. It is the purpose of this article to survey some of these design techniques and their applications.

DATA compression is the conversion of a stream of analog or very high rate discrete data into a stream of relatively low rate data for communication over a digital communication link or storage in a digital memory. As digital communication and secure communication have become increasingly important, the theory and practice of data compression have received increased attention. While it is true that in many systems bandwidth is relatively inexpensive, e.g., fiber optic and cable TV links, in most systems the growing amount of information that users wish to communicate or store necessitates some form of compression for efficient, secure, and reliable use of the communication or storage medium.

A prime example arises with image data, where simple schemes require bit rates too large for many communication links or storage devices. Another example where compression is required results from the fact that if speech is digitized using a simple PCM system consisting of a sampler followed by scalar quantization, the resulting signal will no longer have a small enough bandwidth to fit on ordinary telephone channels. That is, digitization (which may be desirable for security or reliability) causes bandwidth expansion. Hence data compression will be required if the original communication channel is to be used.

The two examples of image compression and speech compression or, as they are often called, image coding and speech coding, are probably the currently most important applications of data compression. They are also among the most interesting for study because experience has shown that both types of data exhibit sufficient structure to permit considerable compression with sufficiently sophisticated codes.

Such conversion of relatively high rate data to lower rate data virtually always entails a loss of fidelity or an increase in distortion. Hence a fundamental goal of data compression is to obtain the best possible fidelity for the given rate or, equivalently, to minimize the rate required for a given fidelity. If a system has a sufficiently high rate constraint, then good fidelity is relatively easy to achieve and techniques such as PCM, transform coding, predictive coding, and adaptive versions of these techniques have become quite popular because of their simplicity and good performance [1, 2, 3]. All of these techniques share a fundamental property: The actual quantization or coding or conversion of continuous quantities into discrete quantities is done on scalars, e.g., on individual real-valued samples of waveforms or pixels of images. PCM does this in a memoryless fashion; that is, each successive input is encoded using a rule that does not depend on any past inputs or outputs of the encoder. Transform coding does it by first taking block transforms of a vector and then scalar coding the coordinates of the transformed vector. Predictive coding does it by quantizing an error term formed as the difference between the new sample and a prediction of the new sample based on past coded outputs.

A fundamental result of Shannon's rate-distortion theory, the branch of information theory devoted to data compression, is that better performance can always be achieved by coding vectors instead of scalars, even if the data source is memoryless, e.g., consists of a sequence of independent random variables, or if the data compression system can have memory, i.e., the action of an encoder at each time is permitted to depend on past encoder inputs or outputs [4, 5, 6, 7, 8]. While some traditional compression schemes such as transform coding operate on vectors and achieve significant improvement over PCM, the quantization is still accomplished on scalars and hence these systems are, in a Shannon sense, inherently suboptimal: better performance is always achievable *in theory* by coding vectors instead of scalars, even if the scalars have been produced by preprocessing the original input data so as to make them uncorrelated or independent!

0740-7467/84/0400-0004$1.00©1984 IEEE

This theory had a limited impact on actual system design because 1) the Shannon theory does not provide constructive design techniques for vector coders, and 2) traditional scalar coders often yield satisfactory performance with enough adaptation and fine tuning. As a result, few design techniques for vector quantizers were considered in the literature prior to the late 1970's when it was found that a simple algorithm of Lloyd [9] for the iterative design of scalar quantization or PCM systems extended in a straightforward way to the design of memoryless vector quantizers, that is, of vector quantizers which encode successive input vectors in a manner not depending on previous encoder input vectors or their coded outputs. Variations of the basic algorithm have since proved useful for the design of vector quantizers with and without memory for a variety of data sources including speech waveforms, speech parameter vectors, images, and several random process models, the latter being useful for gauging the performance of the resulting codes with the optimal performance bounds of information theory.

This paper is intended as a survey of the basic design algorithm and many of its variations and applications. We begin with the simplest example of a memoryless vector quantizer, a vector generalization of PCM. For convenience we use the shorthand VQ for both vector quantization and vector quantizer. Necessary properties of optimal quantizers are described and an algorithm given which uses these properties to iteratively improve a code. For concreteness, we focus on two examples of distortion measures: the ubiquitous mean-squared error and the Itakura-Saito distortion. The first example, which is popular in waveform coding applications, provides a geometric flavor to the development; the second example, which is useful in voice coding applications, helps to demonstrate the generality and power of the technique.

Next, various techniques are described for designing the initial codes required by the algorithm. These techniques also indicate some useful structure that can be imposed on vector quantizers to make them more implementable. Several variations of the basic VQ are described which permit reduced complexity or memory or both at the expense of a hopefully tolerable loss of performance. These include tree-searched codes, product codes, and multistep codes.

We then turn from memoryless vector quantizers to those with memory: feedback vector quantizers such as vector predictive quantizers and finite-state vector quantizers. These codes are not yet well understood, but they possess a structure highly suited to VLSI implementation and initial studies suggest that they offer significant performance gains.

For comparison, we also briefly describe trellis encoding systems or "lookahead" or "delayed decision" or "multipath search" codes which use the same decoder as a feedback vector quantizer but which permit the encoder to base its decision on a longer input data sequence.

A final general code structure is described which uses vector quantization to adapt a waveform coder, which may be another VQ.

We next present a variety of simulation results describing the performance of various VQ systems on various data sources. Examples of all of the above VQ varieties are tested for waveform coding applications on two common data sources: a Gauss Markov source and real sampled speech. One bit per sample coders for these sources are compared on the basis of performance, memory requirements, and computational complexity. Both memoryless and simple feedback vector quantizers are studied for voice coding applications at a rate of 0.062 bits/sample and less and for image coding at a rate of 0.5 bit per sample. One example is given of a simple adaptive predictive vector quantizer for speech waveform coding.

By studying a variety of coding systems on common data sources, the results yield some general comparisons and trends among the various vector quantization techniques. The reader should, however, keep two caveats in mind when interpreting such quantitative results: First, the emphasis here is on low bit rate systems, e.g., speech coders using 1 bit per sample or less and image coders ½ bit per pixel. Comparisons favoring certain systems at such low rates may not be valid for the same systems at higher rates. Second, the numbers reported here are intended to provide comparisons for different systems used on common data sources; they can be compared with other numbers reported in the literature only with great care: the input data and the system design parameters such as sampling rate and pre- or post-filtering may be quite different.

Applications of vector quantization to real data sources such as sampled speech waveforms and images are still young and the algorithms do not yet incorporate the sophisticated "bells and whistles" of many well-established scalar quantization schemes. The preliminary experiments described here, using fairly simple vector quantizers with and without memory, demonstrate that the general approach holds considerable promise for some applications. For example, good quality vocoding systems using VQ and the Itakura-Saito distortion have been developed at 800 bits per second, a significant reduction in the bit rate previously required for comparable quality [10]. While the compression achieved so far in waveform coding and image coding applications using the squared-error distortion has not yet been as significant, we believe that it has yielded comparable or better performance at low rates than traditional scalar schemes of greater complexity. The quality of the ½ bit per pixel images shown here is promising given the simplicity of the coding scheme used.

We attempt to use the minimum of mathematics and a maximum of English in the presentation so as to focus on the intuitive ideas underlying the design and operation of vector quantizers. The detailed descriptions of the various algorithms can be found in the cited references. The reader is also referred to a recent tutorial by Gersho and Cuperman [11] which presents a brief overview of VQ applied to speech waveform coding.

MEMORYLESS VECTOR QUANTIZERS

In this section we introduce the basic definition of memoryless vector quantizers, their properties, and an algorithm for their design.

Quantization

Mathematically, a k-dimensional memoryless vector quantizer or, simply, a VQ (without modifying adjectives) consists of two mappings: an encoder γ which assigns to each input vector $\mathbf{x} = (x_0, x_1, \cdots, x_{k-1})$ a channel symbol $\gamma(\mathbf{x})$ in some channel symbol set $\mathbf{M}$, and a decoder β assigning to each channel symbol v in $\mathbf{M}$ a value in a reproduction alphabet $\hat{\mathbf{A}}$. The channel symbol set is often assumed to be a space of binary vectors for convenience, e.g., $\mathbf{M}$ may be the set of all 2^R binary R-dimensional vectors. The reproduction alphabet may or may not be the same as the input vector space; in particular, it may consist of real vectors of a different dimension.

If $\mathbf{M}$ has M elements, then the quantity $R = \log_2 M$ is called the *rate* of the quantizer in bits per vector and $r = R/k$ is the rate in bits per symbol or, when the input is a sampled waveform, bits per sample.

The application of a quantizer to data compression is depicted in the standard Fig. 1. The input data vectors might be consecutive samples of a waveform, consecutive parameter vectors in a voice coding system, or consecutive rasters or subrasters in an image coding system. For integer values of R it is useful to think of the channel symbols, the encoded input vectors, as binary R–dimensional vectors. As is commonly done in information and communication theory, we assume that the channel is noiseless, that is, that $U_n = \hat{U}_n$. While real channels are rarely noiseless, the joint source and channel coding theorem of information theory implies that a good data compression system designed for a noiseless channel can be combined with a good error correction coding system for a noisy channel in order to produce a complete system. In other words, the assumption of a noiseless channel is made simply to focus on the problem of data compression system design and not to reflect any practical model.

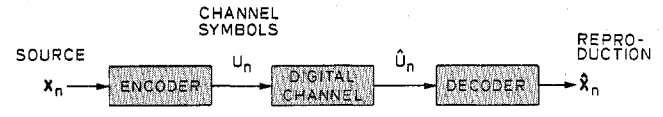

Figure 1. Data Compression System. The data or information source $\{X_n; n = 0, 1, \ldots\}$ is a sequence of random vectors. The encoder produces a sequence of channel symbols $\{U_n; n = 0, 1, 2, \ldots\}$. The sequence $\{\hat{U}_n; n = 0, 1, 2, \ldots\}$ is delivered to the receiver by the digital channel. The decoder then maps this sequence into the final reproduction sequence of vectors $\{\hat{X}_n; n = 0, 1, 2, \ldots\}$.

Observe that unlike scalar quantization, general VQ permits fractional rates in bits per sample. For example, scalar PCM must have a bit rate of at least 1 bit per sample while a k dimensional VQ can have a bit rate of only $1/k$ bits per sample by having only a single binary channel symbol for k-dimensional input vectors.

The goal of such a quantization system is to produce the "best" possible reproduction sequence for a given rate R. To quantify this idea, to define the performance of a quantizer, and to complete the definition of a quantizer, we require the idea of a distortion measure.

Distortion

A distortion measure d is an assignment of a cost $d(\mathbf{x}, \hat{\mathbf{x}})$ of reproducing any input vector $\mathbf{x}$ as a reproduction vector $\hat{\mathbf{x}}$. Given such a distortion measure, we can quantify the performance of a system by an average distortion $Ed(\mathbf{X}, \hat{\mathbf{X}})$ between the input and the final reproduction: A system will be good if it yields a small average distortion. In practice, the important average is the long term sample average or time average

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} d(\mathbf{X}_i, \hat{\mathbf{X}}_i) \tag{1}$$

provided, of course, that the limit makes sense. If the vector process is stationary and ergodic, then, with probability one, the limit exists and equals an expectation $E(d(\mathbf{X}, \hat{\mathbf{X}}))$. For the moment we will assume that such conditions are met and that such long term sample averages are given by expectations. Later remarks will focus on the general assumptions required and their implications for practice.

Ideally a distortion measure should be tractable to permit analysis, computable so that it can be evaluated in real time and used in minimum distortion systems, and subjectively meaningful so that large or small quantitative distortion measures correlate with bad and good subjective quality. Here we do not consider the difficult and controversial issues of selecting a distortion measure; we assume that one has been selected and consider means of designing systems which yield small average distortion. For simplicity and to ease exposition, we focus on two important specific examples:

(1) *The squared error distortion measure:* Here the input and reproduction spaces are k-dimensional Euclidean space

$$d(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|^2 = \sum_{i=0}^{k-1} (x_i - \hat{x}_i)^2,$$

the square of the Euclidean distance between the vectors. This is the simplest distortion measure and the most common for waveform coding. While not subjectively meaningful in many cases, generalizations permitting input-dependent weightings have proved useful and only slightly more complicated. For the squared-error distortion it is common practice to measure the performance of a system by the signal-to-noise ratio (or signal-to-quantization-noise ratio)

$$SNR = 10 \log_{10} \frac{E(\|\mathbf{X}\|^2)}{E[d(\mathbf{X}, \hat{\mathbf{X}})]}.$$

This corresponds to normalizing the average distortion by the average energy and plotting it on a logarithmic scale: Large (small) SNR corresponds to small (large) average distortion.

(2) *The (modified) Itakura-Saito distortion:* This distortion measure is useful in voice coding applications where the receiver is sent a linear model of the underlying voice production process. The distortion measure is based on the "error matching measure" developed in the pioneering work of Itakura and Saito on the PARCOR or LPC approach to voice coding [12]. More generally, this distortion measure is a special case of a minimum relative entropy or discrimination measure; VQ using such distortion measures can be viewed as an application of the minimum relative entropy pattern classification technique introduced by Kullback [13] as an application of information theory to statistical pattern classification. (See also [14, 15].)

We here introduce a minimum of notation to present a definition of the Itakura-Saito distortion measure. Details and generalizations may be found in [16, 17, 14, 15]. Here the input vector can again be considered as a collection of consecutive waveform samples. Now, however, the output vectors have the form $\hat{\mathbf{x}} = (\alpha, a_1, a_2, \cdots, a_p)$, where α is a positive gain or residual energy term and where the a_i with $a_0 = 1$ are inverse filter coefficients in the sense that if

$$A(z) = \sum_{i=0}^{p} a_i z^{-i}$$

then the all-pole filter with z-transform $1/A(z)$ is a stable filter. Here the reproduction vectors may be thought of as all-pole models for synthesizing the reproduction at the receiver using a locally generated noise or periodic source, in other words, as the filter portion of a linear predictive coding (LPC) model in a vocoding (voice coding) system. The Itakura-Saito distortion between the input vector and the model can be defined in the time domain as

$$d(\mathbf{x}, \hat{\mathbf{x}}) = \frac{\mathbf{a}^t \mathbf{R}(\mathbf{x}) \mathbf{a}}{\alpha} - \ln \frac{\alpha_p(\mathbf{x})}{\alpha} - 1,$$

where $\mathbf{a}^t = (1, a_1, \cdots, a_p)$, $\mathbf{R}(\mathbf{x})$ is the $(p+1) \times (p+1)$ sample autocorrelation matrix of the input vector $\mathbf{x}$, and where $\alpha_p(\mathbf{x})$ is an input gain (residual energy) term defined as the minimum value of $\mathbf{b}^t \mathbf{R}(\mathbf{x}) \mathbf{b}$, where the minimum is taken over all vectors $\mathbf{b}$ with first component equal to 1. There are many equivalent forms of the distortion measure, some useful for theory and some for computation. Frequency domain forms show that minimizing the above distortion can be interpreted as trying to match the sample spectrum of the input vector to the power spectral density of the linear all-pole model formed by driving the filter with z-transform $1/A(z)$ by white noise with constant power spectral density $\sqrt{\alpha}$.

The above formula for the distortion is one of the simplest, yet it demonstrates that the distortion measure is indeed complicated—it is not a simple function of an error vector, it is not symmetric in its input and output arguments, and it is not a metric or distance. Because of the intimate connection of this distortion measure with LPC vocoding techniques, we will refer to VQ's designed using this distortion measure as LPC VQ's.

Average distortion

As the average distortion quantifies the performance of a system and since we will be trying to minimize this quantity using good codes, we pause to consider what the average means in theory and in practice.

As previously noted, in practice it is the long term sample average of (1) that we actually measure and which we would like to be small. If the process is stationary and ergodic, then this limiting time average is the same as the mathematical expectation. The mathematical expectation is useful for developing information theoretic performance bounds, but it is often impossible to calculate in practice because the required probability distributions are not known, e.g., there are no noncontroversial generally accepted accurate probability distributions for real speech and image data. Hence a pragmatic approach to system design is to take long sequences of training data, estimate the "true" but unknown expected distortion by the sample average, and attempt to design a code that minimizes the sample average distortion for the training sequence. If the input source is indeed stationary and ergodic, the resulting sample average should be nearly the expected value and the same code used on future data should yield approximately the same averages [18].

The above motivates a training sequence based design for stationary and ergodic data sources. In fact, even if the "true" probability distributions are known as in the case of a Gauss Markov source, the training sequence approach reduces to a standard Monte Carlo approach.

An immediate objection to the above approach, however, is whether or not it makes sense for real sources which may be neither stationary nor ergodic. The answer is an emphatic "yes" in the following sense: The desired property is that if we design a code based on a *sufficiently long* training sequence and then use the code on future data produced by the same source, then the performance of the code on the new data should be roughly that achieved on the training data. The theoretical issue is to provide conditions under which this statement can be made rigorous. For reasonable distortion measures, a sufficient condition for this to be true for memoryless VQ design is that the source be asymptotically mean stationary, it need not be either stationary nor ergodic [19, 20, 21, 22, 23]. Asymptotically mean stationary sources include all stationary sources, block (or cyclo) stationary sources, and asymptotically stationary sources. Processes such as speech which exhibit distinct short term and long term stationarity properties are well modeled by asymp-

totically mean stationary sources [21].

The key point here is that the general design approach using long training sequences does not require either ergodicity nor stationarity to have a solid mathematical foundation. In fact, the mathematics suggest the following pragmatic approach: Try to design a code which minimizes the sample average distortion for a very long training sequence. Then use the code on test sequences produced by the same source, but not in the training sequence. If the performance is reasonably close to the design values, then one can have a certain amount of confidence that the code will continue to yield roughly the same performance in the future. If the training and test performance are significantly different, then probably the training sequence is not sufficiently long. In other words, do not try to prove mathematically that a source is asymptotically mean stationary, instead try to design codes for it and then see if they work on new data.

Henceforth for brevity we will write expectations with the assumption that they are to be interpreted as shorthand for long term sample averages. (A sample average $L^{-1}\sum_{i=0}^{L-1} d(\mathbf{X}_i, \hat{\mathbf{X}}_i)$ is, in fact, an expectation with respect to the sample distribution which assigns a probability of $1/L$ to each vector in the training sequence.)

Properties of optimal quantizers

A VQ is optimal if it minimizes an average distortion $Ed\{\mathbf{X}, \beta[\gamma(\mathbf{X})]\}$. Two necessary conditions for a VQ to be optimal follow easily using the same logic as in Lloyd's [9] classical development for optimal PCM with a mean-squared error distortion measure. The following definition is useful for stating these properties: The collection of possible reproduction vectors $\mathbf{C} = \{\text{all } \mathbf{y} : \mathbf{y} = \beta(v), \text{ some } v \text{ in } \mathbf{M}\}$ is called the *reproduction codebook* or, simply, *codebook* of the quantizer and its members called *codewords* (or templates). The encoder knows the structure of the decoder and hence all of the possible final output codewords.

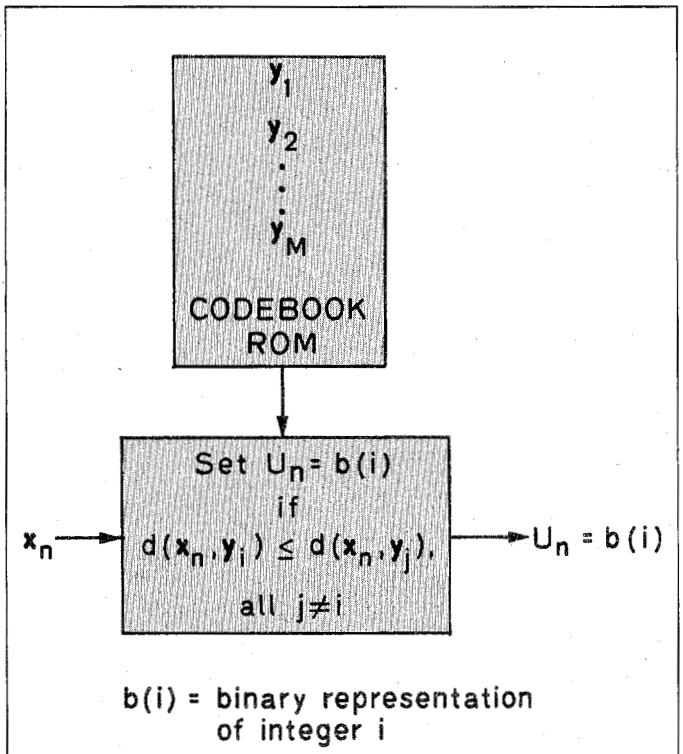

Figure 2. VQ Encoder. The distortion between the input vector and each stored codeword is computed. The encoded output is then the binary representation of the index of the minimum distortion codeword.

Property 1: Given the goal of minimizing the average distortion and given a specific decoder β, no memoryless quantizer encoder can do better than select the codeword v in $\mathbf{M}$ that will yield the minimum possible distortion at the output, that is, to select the channel symbol v yielding the minimum

$$d\{\mathbf{x}, \beta[\gamma(\mathbf{x})]\} = \min_{v \in \mathbf{M}} d[\mathbf{x}, \beta(v)] = \min_{y \in \mathbf{C}} d(\mathbf{x}, y). \quad (2)$$

That is, for a given decoder in a memoryless vector quantizer the best encoder is a minimum distortion or nearest neighbor mapping

$$\gamma(\mathbf{x}) = \min^{-1}_{v \in \mathbf{M}} d[\mathbf{x}, \beta(v)], \quad (3)$$

where the inverse minimum notation means that we select the v giving the minimum of (2).

Gersho [24] calls a quantizer with a minimum distortion encoder a Voronoi quantizer since the Voronoi regions about a set of points in a space correspond to a partition of that space according to the nearest-neighbor rule. The word quantizer, however, is practically always associated with such a minimum distortion mapping. We observe that such a vector quantizer with such a minimum distortion encoder is exactly the Shannon model for a block source code subject to a fidelity criterion which is used in information theory to develop optimal performance bounds for data compression systems.

An encoder γ can be thought of as a partition of the input space into cells where all input vectors yielding a common reproduction are grouped together. Such a partition according to a minimum distortion rule is called a Voronoi or Dirichlet partition. A general minimum distance VQ encoder is depicted In Fig. 2.

A simple example of such a partition and hence of an encoder is depicted in Fig. 3 (a more interesting example follows shortly). Observe that this vector quantizer is just two uses of a scalar quantizer in disguise.

As the minimum distortion rule optimizes the encoder of a memoryless VQ for a decoder, we can also optimize the decoder for a given encoder.

Property 2: Given an encoder γ, then no decoder can do better than that which assigns to each channel symbol v the generalized centroid (or center of gravity or barycenter) of all source vectors encoded into v, that is,

$$\beta(v) = cent(v) = \min^{-1}_{\hat{\mathbf{x}} \in \hat{\mathbf{A}}} E(d(\mathbf{X}, \hat{\mathbf{x}}) \,|\, \gamma(\mathbf{X}) = v), \quad (4)$$

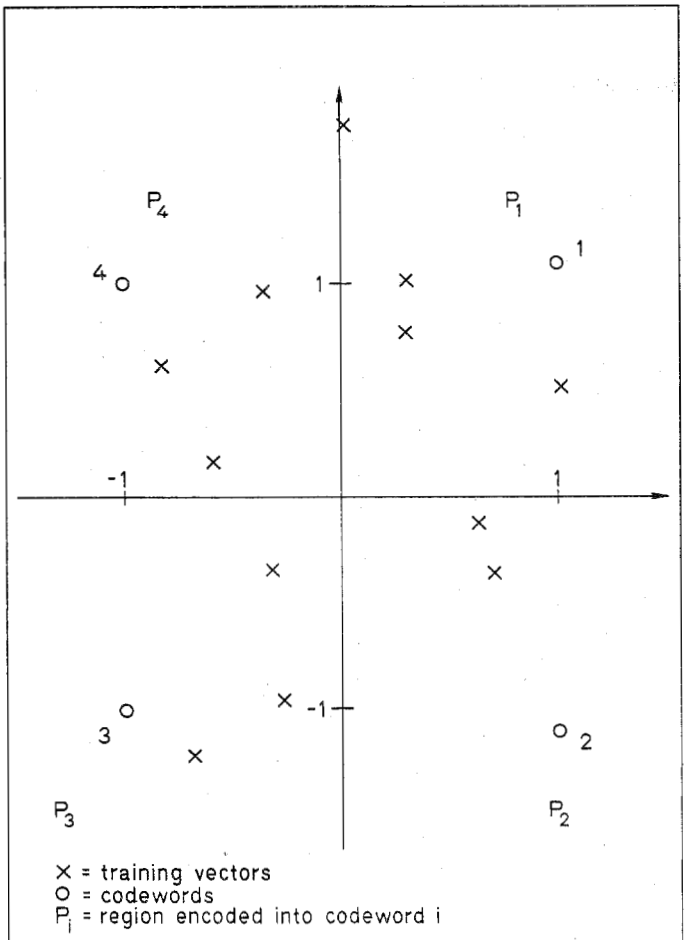

Figure 3. Two-Dimensional Minimum Distortion Partition. The four circles are the codewords of a two-dimensional codebook. The Voronoi regions are the quadrants containing the circles. The *x*'s were produced by a training sequence of twelve two-dimensional Gaussian vectors. Each input vector is mapped into the nearest-neighbor codeword, that is, the circle in the same quadrant.

that is, $\beta(v)$ is the vector yielding the minimum conditional average distortion given that the input vector was mapped into v.

While minimizing such a conditional average may be quite difficult for an arbitrary random process and distortion measure, it is often easy to find for a sample distribution and a nice distortion measure. For example, the centroid in the case of a sample distribution and a squared-error distortion measure is simply the ordinary Euclidean centroid or the vector sum of all input vectors encoded into the given channel symbol, that is, given the sample distribution defined by a training sequence $\{\mathbf{x}_i;\ i = 0, 1, \ldots, L-1\}$, then

$$cent(v) = \frac{1}{i(v)} \sum_{\mathbf{x}_i:\ \gamma(\mathbf{x}_i)=v} x_i,$$

where $i(v)$ is the number of indices i for which $\gamma(\mathbf{x}_i) = v$.

The Euclidean centroids of the example of Fig. 3 are depicted in Fig. 4. (The numerical values may be found in [25].) The new codewords better represent the training vectors mapping into the old codewords, but they yield a different minimum distortion partition of the input alphabet, as indicated by the broken line in Fig. 3. This is the key of the algorithm: iteratively optimize the codebook for the old encoder and then use a minimum distortion encoder for the new codebook.

The Itakura-Saito distortion example is somewhat more complicated, but still easily computable. As with the squared error distortion, one groups all input vectors yielding a common channel symbol. Instead of averaging the vectors, however, the sample autocorrelation matrices for all of the vectors are averaged. The centroid is then given by the standard LPC all-pole model for this average autocorrelation, that is, the centroid is found by a standard Levinson's recursion run on the average autocorrelation.

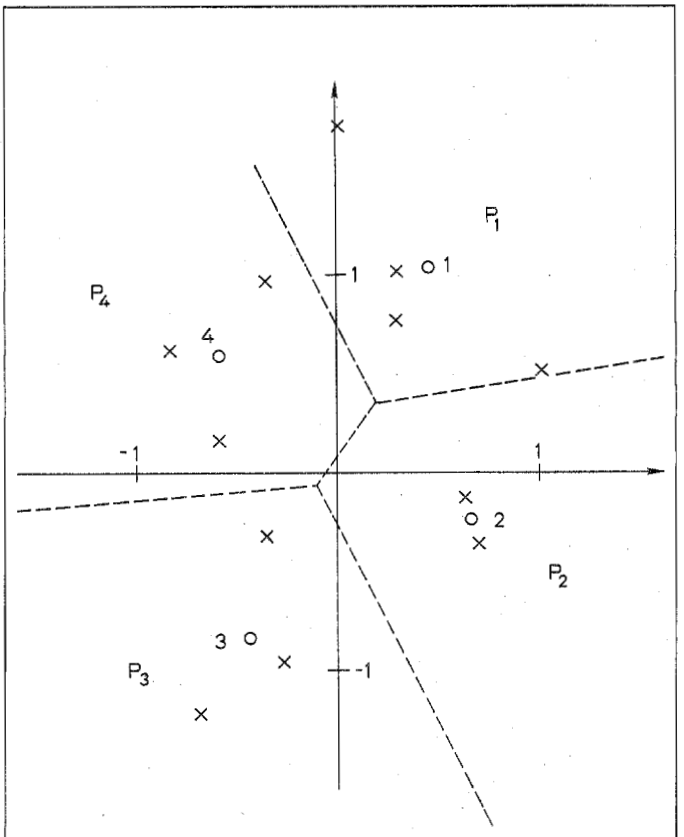

Figure 4. Centroids of Figure 3. The new centroids of the old Voronoi regions of Fig. 3 are drawn as circles. Note that the centroid computation has moved the codewords to better represent the input vectors which yielded those codewords, that is, if one used the same encoder (as in Fig. 3), but replaced the reproduction codewords produced at the decoder by these new centroids, the average distortion would decrease. The broken line delineates the new Voronoi regions for these codewords.

The generalized Lloyd algorithm

The fact that the encoder can be optimized for the decoder and vice versa formed the basis of Lloyd's original optimal PCM design algorithm for a scalar random variable with a known probability density function and a squared error distortion. The general VQ design algorithms considered here are based on the simple observation that Lloyd's basic development is valid for vectors, for sample distributions, and for a variety of distortion measures. The only requirement on the distortion measure is that one can compute the centroids. The basic algorithm is the following:

Step 0. Given: A training sequence and an initial decoder.

Step 1. Encode the training sequence into a sequence of channel symbols using the given decoder minimum distortion rule. If the average distortion is small enough, quit.

Step 2. Replace the old reproduction codeword of the decoder for each channel symbol v by the centroid of all training vectors which mapped into v in Step 1. Go to Step 1.

Means of generating initial decoders will be considered in the next section. Each step of the algorithm must either reduce average distortion or leave it unchanged. The algorithm is usually stopped when the relative distortion decrease falls below some small threshold. The algorithm was developed for vector quantizers, training sequences, and general distortion measures by Linde, Buzo, and Gray [25] and it is sometimes referred to as the LBG algorithm. Previously Lloyd's algorithm had been considered for vectors and difference distortion measures in cluster analysis and pattern recognition problems (e.g., MacQueen [26] and Diday and Simon [27]) and in two-dimensional quantization (e.g., Chen [28] and Adoul *et al.* [29]). Only recently, however, has it been extensively studied for vector quantization applications using several different distortion measures.

Before continuing, it should be emphasized that such iterative improvement algorithms need not in general yield truly optimum codes. It is known that subject to some mathematical conditions the algorithm will yield locally optimum quantizers, but in general there may be numerous such codes and many may yield poor performance. (See, e.g., [30].) It is often useful, therefore, to enhance the algorithm's potential by providing it with good initial codebooks and perhaps by trying it on several different initial codebooks.

INITIAL CODEBOOKS

The basic design algorithm of the previous section is an iterative improvement algorithm and requires an initial code to improve. Two basic approaches have been developed: One can start with some simple codebook of the correct size or one can start with a simple small codebook and recursively construct larger ones.

"Random" codes

Perhaps the simplest example of the first technique is that used in the k-means variation of the algorithm [26]: Use the first 2^R vectors in the training sequence as the initial codebook. An obvious modification more natural for highly correlated data is to select several widely spaced words from the training sequence. This approach is sometimes called random code generation, but we avoid this nomenclature because of its confusion with the random code techniques of information theory which are used to prove the performance bounds.

Product codes

Another example of the first approach is to use a scalar code such as a uniform quantizer k times in succession and then prune the resulting vector codebook down to the correct size. The mathematical model for such a code is a product code, which we pause to define for current and later use: Say we have a collection of codebooks C_i, $i = 0,1,\ldots,m-1$, each consisting of M_i vectors of dimension k_i and having rate $R_i = \log_2 M_i$ bits per vector. Then the *product codebook* C is defined as the collection of all $M = \Pi_i M_i$ possible concatenations of m words drawn successively from the m codebooks C_i. The dimension of the product codebook is $k = \sum_{i=0}^{m-1} k_i$, the sum of the dimensions of the component codebooks. The product code is denoted mathematically as a Cartesian product:

$$C = \overset{m-1}{\underset{i=0}{\times}} C_i = \{\textit{all vectors of the form } (\hat{x}_0, \hat{x}_1, \cdots, \hat{x}_{m-1}); \hat{x}_i \textit{ in } C_i;\ i = 0,1,\ldots,m-1\}$$

Thus, for example, using a scalar quantizer with rate R/k k times in succession yields a product k-dimensional vector quantizer of rate R bits per vector. This product code can be used as an initial code for the design algorithm. The scalar quantizers may be identical uniform quantizers with a range selected to match the source, or they may be different, e.g., a positive codebook for a gain and uniform quantizers for $[-1,1]$ for reflection coefficients in an LPC VQ system.

In waveform coding applications where the reproduction and input alphabets are the same—k-dimensional Euclidean space—an alternative product code provides a means of growing better initial guesses from smaller dimensional codes [31]. Begin with a scalar quantizer C_0 and use a two-dimensional product code $C_0 \times C_0$ as an initial guess for designing a two-dimensional VQ. On completion of the design we have a two-dimensional code, say C^2. Form an initial guess for a three dimensional code as all possible pairs from C^2 and scalars from C_0, that is, use the product code $C^2 \times C_0$ as an initial guess. Continuing in this way, given a good $k-1$ dimensional VQ described by a codebook C^{k-1}, an initial guess for a k-dimensional code design is the product code $C^{k-1} \times C_0$. One can also use such product code constructions with a different initial scalar code C_0, such as those produced by the scalar version of the next algorithm.

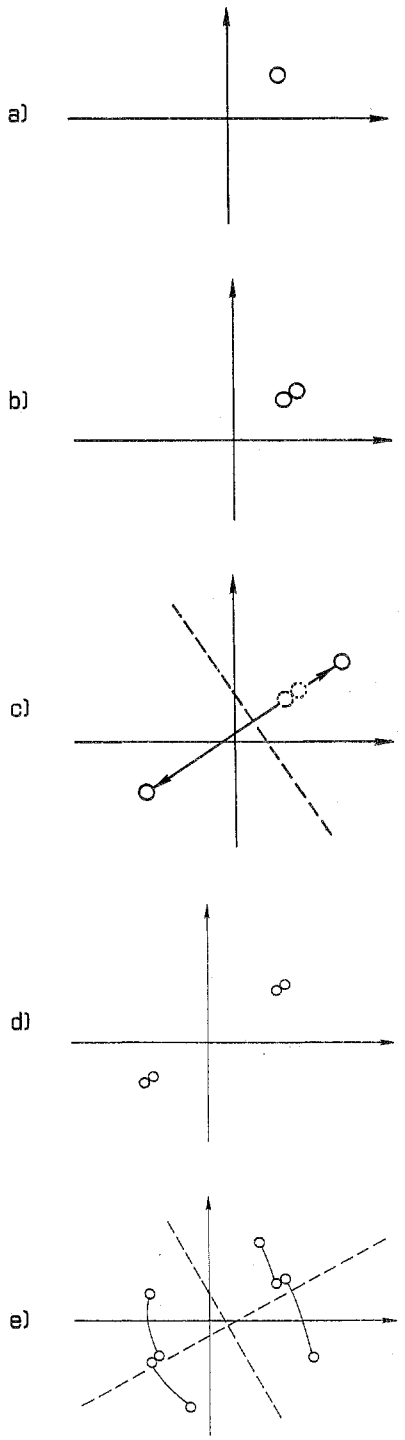

Figure 5. Splitting. A large code is defined in stages: at each stage each codeword of a small code is split into two new codewords, giving an initial codebook of twice the size. The algorithm is run to get a new better codebook. (a) Rate 0: The centroid of the entire training sequence. (b) Initial Rate 1: The one codeword is split to form an initial guess for a two word code. (c) Final Rate 1: The algorithm produces a good code with two words. The dotted line indicates the Voronoi regions. (d) Initial Rate 2: The two words are split to form an initial guess for a four word code. (e) Final Rate 2: The algorithm is run to produce a final four word code.

Splitting

Instead of constructing long codes from smaller dimensional codes, we can construct a sequence of bigger codes having a fixed dimension using a "splitting" technique [25, 16]. This method can be used for any fixed dimension, including scalar codes. Here one first finds the optimum 0 rate code—the centroid of the entire training sequence, as depicted in Fig. 5a for a two-dimensional input alphabet. This single codeword is then split to form two codewords (Fig. 5b). For example, the energy can be perturbed slightly to form a second distinct word or one might purposefully find a word distant from the first. It is convenient to have the original codeword a member of the new pair to ensure that the distortion will not increase. The algorithm is then run to get a good rate 1 bit per vector code as indicated in Fig. 5c. The design continues in this way in stages as shown: the final code of one stage is split to form an initial code for the next.

VARIATIONS OF MEMORYLESS VECTOR QUANTIZERS

In this section we consider some of the variations of memoryless vector quantization aimed at reducing the computation or memory requirements of a full search memoryless VQ.

Tree-searched VQ

Tree-searched vector quantizers were first proposed by Buzo *et al.* [16] and are a natural byproduct of the splitting algorithm for generating initial code guesses. We focus on the case of a binary tree for simplicity, but more general trees will provide better performance while retaining a significant reduction in complexity.

Say that we have a good rate 1 code as in Fig. 5c and we form a new rate two code by splitting the two codewords as in Fig. 5d. Instead of running a full search VQ design on the resulting 4-word codebook, however, we divide the training sequence into two pieces, collecting together all those vectors encoded into a common word in the 1 bit codebook, that is, all of the training sequence vectors in a common cell of the Voronoi partition. For each of these subsequences of training vectors, we then find a good 1-bit code using the algorithm. The final codebook (so far) consists of the four codewords in the two 1-bit codebooks designed for the two subsequences. A tree-searched encoder selects one of the words not by an ordinary full search of this codebook, but instead it uses the first one bit codebook designed on the whole sequence to select a second code and it then picks the best word in the second code. This encoder can then be used to further subdivide the training sequence and construct even better codebooks for the subsequences. The encoder operation can be depicted as a tree in Fig. 6.

The tree is designed one layer at a time; each new layer being designed so that the new codebook available from each node is good for the vectors encoded into the node. Observe that there are 2^R possible reproduction vectors as in the full search VQ, but now R binary searches are made instead of a single 2^R-ary search. In addition, the encoder

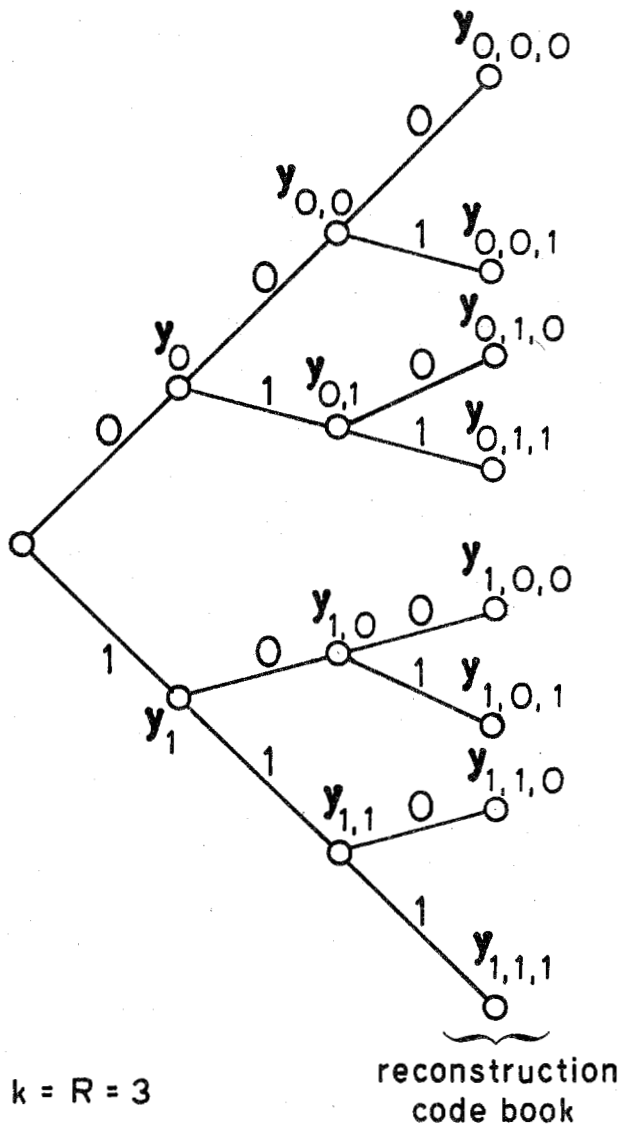

Figure 6. Tree-Searched VQ. A binary encoder tree is shown for a three-dimensional one bit per sample VQ. The encoder makes a succession of R minimum distortion choices from binary codebooks, where the available codebook at each level consists of labels of the nodes in the next level. The labels of the nodes of the final layer are the actual reproduction codewords. At each node the encoder chooses the minimum distortion available label and, if the new index is a 0 (1), sends a channel symbol of 0 (1) and advances up (down) to the next node. After R binary selections the complete channel codeword has been sent and the reproduction codeword specified to the decoder.

storage requirements have doubled. The encoder is no longer optimal for the decoder in the sense of Property 1 since it no longer can perform an exhaustive search of the codebook. The search, however, is much more efficient if done sequentially than is a full search. Thus one may trade performance for efficiency of implementation.

Nonbinary trees can also be used where at the i^{th} layer codebooks of rate R_i are used and the overall rate is then $\Sigma_i R_i$. For example, a depth three tree for VQ of LPC parameter vectors using successive rates of 4, 4, and 2 bits per vector yields performance nearly as good as a full search VQ of the same total rate of 10 bits per vector, yet for the tree search one need only compute $2^4 + 2^4 + 2^2 = 36$ distortions instead of $2^{10} = 1028$ distortions [10].

Other techniques can be used to design tree-searched codes. For example, Adoul *et al.* [32] use a separating hyperplane approach. Another approach is to begin with a full search codebook and to design a tree-search into the codebook. One technique for accomplishing this is to first group the codewords into close disjoint pairs and then form the centroids of the pairs as the node label of the immediate ancestor of the pair. One then works backwards through the tree, always grouping close pairs. Ideally, one would like a general design technique for obtaining a tree search into an arbitrary VQ codebook with only a small loss of average distortion. Gersho and Cheng [33] have reported preliminary results for designing a variable-length tree search for an arbitrary codebook and have demonstrated its implementability for several small dimensional examples.

Multistep VQ

A multistep VQ is a tree-searched VQ where only a single small codebook is stored for each layer of the tree instead of a different codebook for each node of each layer. Such codes provide the computation reduction of tree-searched codes while reducing the storage requirements below that of even ordinary VQ's. The first example of such a code was the multistage codebook [34]. For simplicity we again confine interest to codes which make a sequence of binary decisions. The first layer binary code is designed as in the tree-searched case. This codebook is used to encode the training sequence and then a training sequence of error or residual vectors is formed. For waveform coding applications the error vectors are simply the difference of the input vectors and their codewords. For vocoding applications, the error vectors are residuals formed by passing the input waveform through the inverse filter $A(z)/\alpha$. The algorithm is then run to design a binary VQ for this vector training sequence of coding errors. The reconstruction for these two bits is then formed by combining the two codewords: For waveform coding this is accomplished by adding the first codeword to the error codeword. For voice coding this is accomplished by using the cascade of two all-pole filters for synthesis. This reproduction can then be used to form a "finer" error vector and a code designed for it. Thus an input vector is encoded in stages as with the tree-searched code, but now only R binary codebooks and hence $2R$ total codewords need to be stored. Observe that there are still 2^R possible final codewords, but we have not needed this much storage because the code can be constructed by adding different combinations of a smaller set of words. A multistage VQ is depicted in Fig. 7.

Product codes

Another useful structure for a memoryless VQ is a prod-

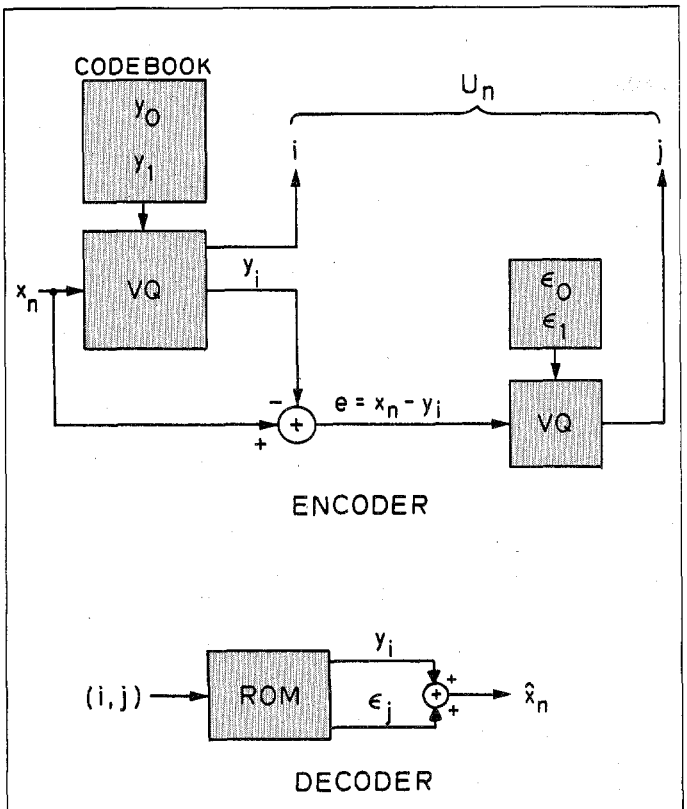

Figure 7. Multistage VQ with 2 Stages. The input vector is first encoded by one VQ and an error vector is formed. The second VQ then encodes the error vector. The two channel symbols from the two VQ's together form the complete channel symbol for the entire encoder. The decoder adds together the corresponding reproduction vectors.

uct code. In one extreme, multiple use of scalar quantizers is equivalent to product VQ's and are obviously simple to implement. More general product VQ's, however, may permit one to take advantage of the performance achievable by VQ's while still being able to achieve the higher rates required for good fidelity. In addition, such codes may yield a smaller computational complexity than an ordinary VQ of the same rate and performance (but different dimension). The basic technique is useful when there are differing aspects of the input vector that one might wish to code separately because of different effects, e.g., on dynamic range or finite word length implementation.

Gain/shape VQ

One example of a product code is a gain/shape VQ where separate, but interdependent, codes are used to code the "shape" and "gain" of the waveform, where the "shape" is defined as the original input vector normalized by removal of a "gain" term such as energy in a waveform coder or LPC residual energy in a vocoder. Gain/shape encoders were introduced by Buzo *et al.* [16] and were subsequently extended and optimized by Sabin and Gray [35, 36]. A gain/shape VQ for waveform coding with a squared-error distortion is illustrated in Fig. 8.

Figure 8 sketches the surprising fact that for the squared error case considered, the two-step selection of the product codeword is an optimal encoding for the given product codebook. We emphasize that here the encoder is optimal for the given product codebook or decoder, but the codebook itself is in general suboptimal because of the constrained product form. A similar property holds for the Itakura-Saito distortion gain/shape VQ. Thus in this case if one devotes R_s bits to the shape and R_g bits to the gain, where $R_s + R_g = R$, then one need only compute 2^{R_s} vector distortions and an easy scalar quantization. The full search encoder would require 2^R vector distortions, yet both encoders yield the same minimum distortion codeword!

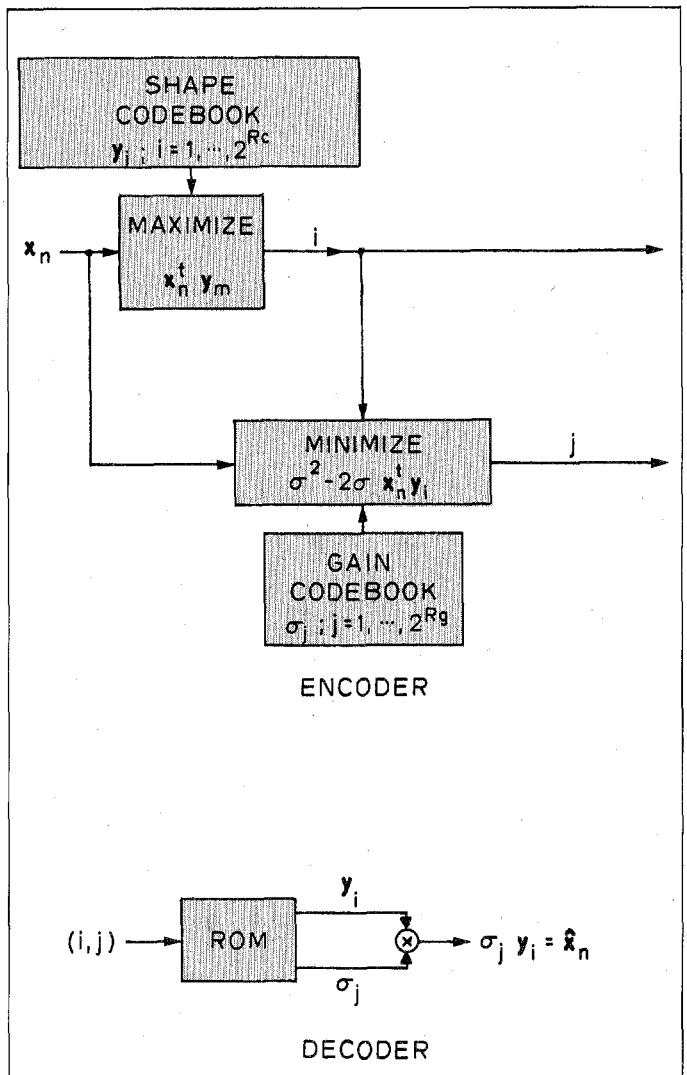

Figure 8. Gain/Shape VQ. First a unit energy shape vector is chosen to match the input vector by maximizing the inner product over the codewords. Given the resulting shape vector, a scalar gain codeword is selected so as to minimize the indicated quantity. The encoder yields the product codeword $\sigma_j y_i$ with the minimum possible squared error distortion from the input vector. Thus this multistep encoder is optimum for the product codebook.

Variations of the basic VQ algorithm can be used to iteratively improve a gain shape code by alternately optimizing the shape for the gain and vice versa. The resulting conditional centroids are easy to compute. The centroid updates can be made either simultaneously or alternately after each iteration [36].

One can experimentally determine the optimal bit allocation between the gain and the shape codebooks.

Separating mean VQ

Another example of a multistep product code is the separating mean VQ where a sample mean instead of an energy term is removed [37]. Define the sample mean $\langle \mathbf{x} \rangle$ of a k-dimensional vector by $k^{-1} \sum_{i=0}^{k-1} x_i$. In a separated mean VQ one first uses a scalar quantizer to code the sample mean of a vector, then the coded sample mean is subtracted from all of the components of the input vector to form a new vector with approximately zero sample mean. This new vector is then vector quantized. Such a system is depicted in Fig. 9. The basic motivation here is that in image coding the sample mean of pixel intensities in a small rectangular block represents a relatively slowly varying average background value of pixel intensity around which there are variations.

To design such a VQ, first use the algorithm to design a scalar quantizer for the sample mean sequence $\langle \mathbf{x}_j \rangle$, $j = 0, 1, \ldots, L-1$. Let $q(\langle \mathbf{x} \rangle)$ denote the reproduction for $\langle \mathbf{x} \rangle$ using the quantizer. Then use the vector training sequence $\mathbf{x}_j - q(\langle \mathbf{x}_j \rangle)\mathbf{1}$, where $\mathbf{1} = (1, 1, \ldots, 1)$, to design a VQ for the difference. Like the gain/shape VQ, a product codebook and a multistep encoder are used, but unlike the gain/shape VQ it can be shown that the multistep encoder here does not select the best possible mean, shape pair, that is, the multistep encoder is not equivalent to a full search encoder.

Lattice VQ

A final VQ structure capable of efficient searches and memory usage is the lattice quantizer, a k-dimensional generalization of the scalar uniform quantizer. A lattice in k-dimensional space is a collection of all vectors of the form $\mathbf{y} = \sum_{i=0}^{n-1} a_i \mathbf{e}_i$, where $n \leq k$, where $\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}$ are a set of linearly independent vectors in $\mathbf{R}^k$, and where the a_i are arbitrary integers. A lattice quantizer is a quantizer whose codewords form a subset of a lattice. Lattice quantizers were introduced by Gersho [38] and the performance and efficient coding algorithms were developed for many particular lattices by Conway and Sloane [39,40,41] and Barnes and Sloane [42]. The disadvantage of lattice quantizers is that they cannot be improved by a variation of the Lloyd algorithm without losing their structure and good quantizers produced by the Lloyd algorithm cannot generally be well approximated by lattices. Lattice codes can work well on source distributions that are approximately uniform over a bounded region of space. In fact, lattices that are asymptotically optimal in the limit of large rate are known for this case in two and three dimensions and good lattices are known for dimensions up to 16. Ideally, one would like to take a full search, unconstrained VQ and find some fast means of encoding having complexity more like the above techniques than that of the full search. For example, some form of multidimensional companding followed by a lattice quantizer as suggested by Gersho [24] would provide both good performance and efficient implementation. Unfortunately, however, no design methods accomplishing this goal have yet been found.

FEEDBACK VECTOR QUANTIZERS

Memory can be incorporated into a vector quantizer in a simple manner by using different codebooks for each input vector, where the codebooks are chosen based on past input vectors. The decoder must know which codebook is being used by the encoder in order to decode the channel symbols. This can be accomplished in two ways: 1) The encoder can use a codebook selection procedure that depends only on past encoder outputs and hence the codebook sequence can be tracked by the decoder. 2) The decoder is informed of the selected codebook via a special low-rate side channel. The first approach is called feedback vector quantization and is the

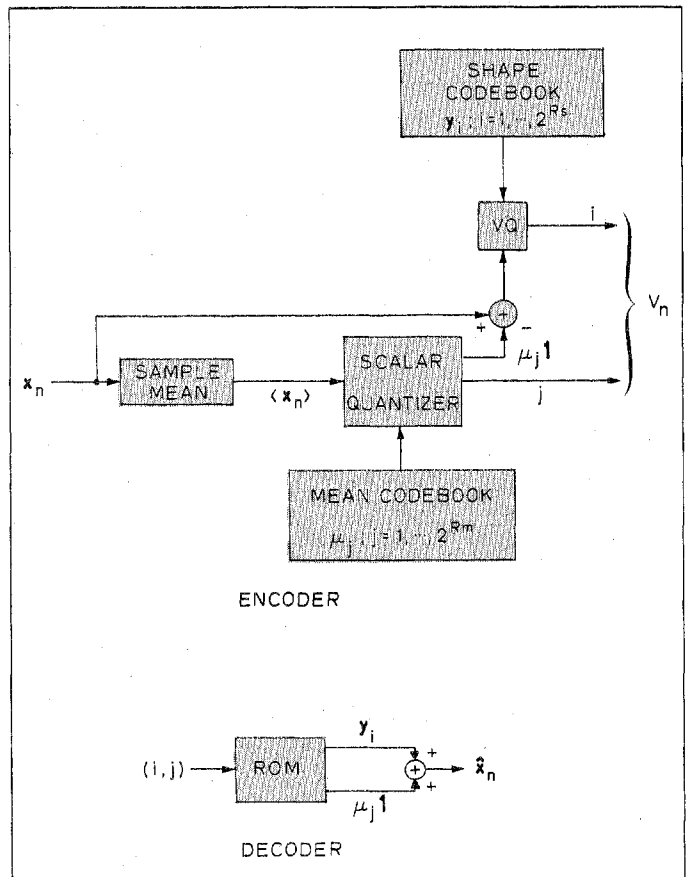

Figure 9. Separating Mean VQ. The sample mean of the input vector is computed, scalar quantized, and then subtracted from each component of the input vector. The resulting vector with approximately zero sample mean is then vector quantized. The decoder adds the coded sample mean to all components of the coded shape vector.