Chapter 11

Vector Quantization II: Optimality and Design

11.1 Introduction

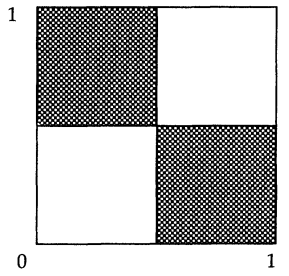
We now turn to the optimality properties of vector quantizers, the vector extensions of the properties developed in Chapter 6. Such properties aid in the design of good quantizers by giving simple conditions that a quantizer must satisfy if it is to be optimal and by giving a simple iterative technique for improving a given quantizer. This technique yields the generalized Lloyd algorithm for codebook design, the most widely used method for designing good vector quantizers. We also explore the additional structure of a class of quantizers that is particularly amenable to low complexity implementation: polytopal quantizers. Unlike Chapter 6 the approximations to the optimal performance for high resolution quantizers are not explored in depth. A few of the ideas are sketched, but the detailed developments are left to the literature. See in particular Chapter 5 of [159] and the original references such as [347], [348], [349], [137], [345], [38].

Statistical Advantages of Vector Quantization

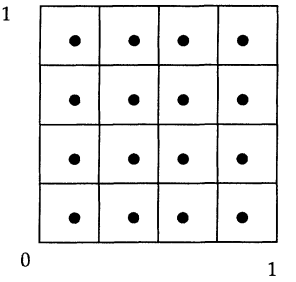
Before proceeding with the topic of optimality, it is helpful to gain some initial insight into the superiority of VQ over scalar quantization for random input vectors. In Chapter 10, we noted the generality of VQ as shown in Theorem 10.1.1 as is evident from the increased freedom in choosing the partition geometry for VQ compared to the very restrictive geometry in the case where each vector component is scalar quantized and the resulting quantization cells are rectangles. These results indicate the greater

flexibility of VQ and make very clear that scalar quantization is simply a restricted special case of VQ. Now we wish to go further and demonstrate that indeed VQ can offer a performance gain that is tied to the degree of correlation between the vector components. At this point, the arguments will be intuitive and informal. Later, after we formulate statistical measures of performance for VQ, we can be more explicit in demonstrating the advantage of VQ over scalar quantization.

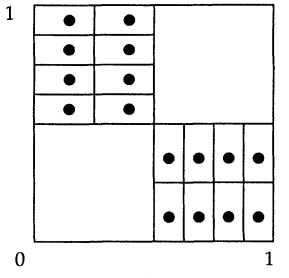
Consider the two-dimensional random variable $\mathbf{X} = (X_1, X_2)$, which has pdf uniformly distributed over the shaded regions within the unit square as shown in Figure 11.1(a).



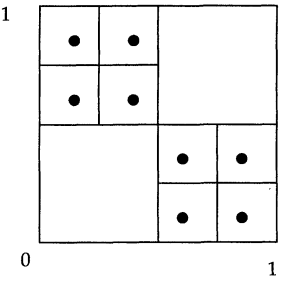
(a) Two-dimensional pdf



(b) Partition by scalar quantization



(c) Partition by VQ



(d) Partition by alternative VQ

Figure 11.1: Two-dimensional quantization

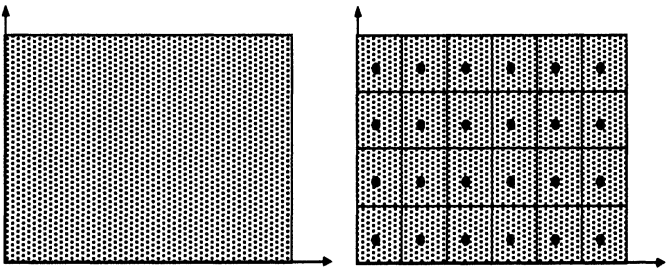
X_1 and X_2 each has a marginal distribution that is uniform over the interval $(0, 1)$. Notice that the two-component random variables are statistically dependent since the probability that $X_1 < 0.5$ conditioned on $X_2 = \alpha$ depends very strongly on whether α is less than or greater than 0.5. Since each component is uniformly distributed, scalar quantization of each component is optimal for a uniform quantizer. Figure 11.1(b) shows

a partition of the domain of $\mathbf{X}$ based on individual quantization of the components in an optimal manner (uniform step size in this case) into 16 cells, whereas Figure 11.1(c) shows another partition into 16 cells available if VQ is used rather than scalar quantization of individual components. Clearly, the latter partition will be superior to the former for any reasonable choice of performance measure. In fact, it is easy to see that the worst-case quantization error measured by Euclidean distance between $\mathbf{X}$ and $Q(\mathbf{X})$ is $\sqrt{2}/8 \approx 0.18$ in case (b) and is $\sqrt{5}/16 \approx 0.13$ in case (c). The total number of bits (4) is the same in both cases. Thus, in this example, VQ is clearly superior.

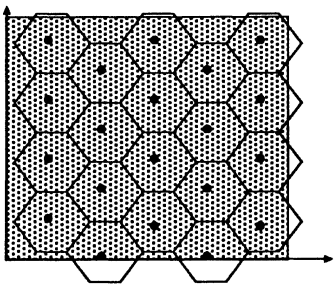
It is easy to extend this intuitive idea to recognize that whenever some correlation exists between the components of a vector, some performance gain can be expected with VQ and stronger correlation produces greater gain. Another feature of this example is that the statistical dependence includes a *linear* dependence (correlation), but it is not limited to a linear dependence. A linear transformation cannot be found to map (X_1, X_2) into a pair of statistically independent random variables. VQ has the appealing feature that it efficiently quantizes vectors which have *nonlinear* statistical dependence between the components.

On the other hand, if the dependence between two random variables were purely linear (as in the case of a joint Gaussian pdf), we would be able to "decorrelate" them by a linear transformation leading to two new independent random variables. This might suggest that VQ has nothing to offer in cases of purely linear dependence. A linear transformation followed by scalar quantization of the components would appear to be optimal in this case. Not so! This is a common misconception; in fact, this coding technique, which is simply transform coding [186] as described in Chapter 8, is theoretically *suboptimal* even in the case of a Gaussian vector. The fallacy lies in the assumption that nothing is to be gained by applying VQ to statistically independent random variables.

In fact, VQ can indeed give superior performance over scalar quantization even when the components of a random vector are statistically independent of each other! Although this follows from the basic Shannon source coding theorems [134],[32],[159], it is also readily seen by considering the following example. Suppose that the two-dimensional vector $\mathbf{X}$ is uniformly distributed on the square domain shown in Figure 11.2(a). In Figure 11.2(b), a rectangular partition is shown corresponding to scalar quantization of each component. For the example to work we assume that the number of quantization points is large, even though only a modest number are shown in the figure to avoid clutter. In Figure 11.2(c), a hexagonal partition is shown. Except possibly at the edge regions, it is easily verified that the hexagonal partition has a lower worst-case error based on Euclidean



(a) Two-Dimensional Region (b) Rectangular (Scalar) Region



(c) Hexagonal Partition

Figure 11.2: Two-Dimensional Quantization

distance than the square partition, for the same number of partition cells. It can be shown that for a statistical error criterion (average squared Euclidean distance) the hexagonal partition performs better than does the rectangular partition if the edge effects are assumed negligible. The point here is that by using a vector quantizer, one can choose cells whose shape more efficiently fills space than do the multidimensional rectangles forced by scalar quantization. These potential advantages of VQ:

- the ability to exploit linear and nonlinear dependence among the vector coordinates and
- the extra freedom in choosing the multidimensional quantizer cell shapes

are discussed in depth from an intuitive point of view in Makhoul et al. [229] and from a more quantitative point of view using multidimensional high rate quantizer theory in Lookabaugh et al. [221], where one can weigh the relative gains due to each advantage.

Another advantage offered by VQ is the flexibility of codebook sizes N that can be used. If scalar quantization of the components of a k-

dimensional vector uses N_1 levels, then the codebook size of the equivalent vector quantizer is N_1^k. On the other hand, direct use of VQ on the vector allows arbitrary partitions with any integer number N of cells. Thus in the example of Figure 11.1, the random vector could be coded with $N = 8$ cells as shown in Figure 11.1(d). Note that the maximum error achieved with this 8-point vector quantizer is exactly the same as in the case of the 16-point rectangular quantizer of Figure 11.1(b). The resolution in case (b) is 2 bits/component whereas it is $(\log_2 8)/2 = 1.5$ bits/component in case (c). An important advantage of VQ over scalar quantization is that VQ can achieve *fractional* values of resolution (measured in bits per sample or bits per vector component). This feature is particularly important where low resolution is required due to limited bit allocations for low bit-rate applications.

11.2 Optimality Conditions for VQ

The principal goal in design of vector quantizers is to find a codebook, specifying the decoder, and a partition or encoding rule, specifying the encoder, that will maximize an overall measure of performance considering the entire sequence of vectors to be encoded over the lifetime of the quantizer. The overall performance can be assessed by either a *statistical average* of a suitable distortion measure or by a *worst-case* value of distortion. We focus here only on statistical criteria. The statistical average of the distortion for a vector quantizer $Q(\cdot)$, can be expressed as

$$D = Ed(\mathbf{X}, Q(\mathbf{X})) = \int d(\mathbf{x}, Q(\mathbf{x}))f_{\mathbf{X}}(\mathbf{x})\,d\mathbf{x}, \tag{11.2.1}$$

where $f_{\mathbf{X}}(\mathbf{x})$ is the (joint) pdf of the vector $\mathbf{X}$ and the integration above is understood to be a multiple integral over the k-dimensional space. When the input vector has a *discrete* distribution, a case that will also be of interest to us later, it is more convenient to avoid the use of a pdf made up of delta functions and instead describe the distribution by the probability mass function, (pmf), denoted by $p_{\mathbf{X}}(\mathbf{x})$. Then we have

$$D = Ed(\mathbf{X}, Q(\mathbf{X})) = \sum_i d(\mathbf{x}_i, Q(\mathbf{x}_i))p_{\mathbf{X}}(\mathbf{x}_i), \tag{11.2.2}$$

where $\{\mathbf{x}_i\}$ are the values of $\mathbf{X}$ that have nonzero probability.

We assume that the codebook size N is given, the k-dimensional input random vector, $\mathbf{X}$, is statistically specified, and a particular distortion measure $d(\cdot, \cdot)$ has been selected. We wish to determine the necessary conditions for a quantizer to be optimal in the sense that it minimizes the

average distortion for the given conditions. As in the scalar case, we proceed by finding the necessary condition for the encoder to be optimal for a given decoder. Then for a given encoder we find the necessary condition for the decoder to be optimal. Recall that the encoder is completely specified by the partition of $\mathcal{R}^k$ into the cells $R_1, R_2, \cdots, R_N$, and the decoder is completely specified by the codebook, $\mathcal{C} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_N\}$. The optimality conditions presented next will explicitly determine the optimal partition for a given codebook and the optimal codebook for a given partition.

We first consider the optimization of the encoder for a fixed decoder. For a given codebook, an optimal partition is one satisfying the *nearest neighbor condition* (NN condition): for each i, all input points closer to code vector $\mathbf{y}_i$ than to any other code vector should be assigned to region R_i. Thus:

Nearest Neighbor Condition

For a given set of output levels, $\mathcal{C}$, the optimal partition cells satisfy

$$R_i \subset \{\mathbf{x} : d(\mathbf{x}, \mathbf{y}_i) \leq d(\mathbf{x}, \mathbf{y}_j); \text{ for all } j\}, \tag{11.2.3}$$

that is,

$$Q(x) = \mathbf{y}_i \text{ only if } d(\mathbf{x}, \mathbf{y}_i) \leq d(\mathbf{x}, \mathbf{y}_j) \text{ all } j.$$

Thus, given the decoder, the encoder is a minimum distortion or nearest neighbor mapping, and hence

$$d(\mathbf{x}, Q(\mathbf{x})) = \min_{\mathbf{y}_i \in \mathcal{C}} d(\mathbf{x}, \mathbf{y}_i). \tag{11.2.4}$$

The astute reader will note that the condition is the same as that for scalar quantizers. The condition states that if $\mathbf{x}$ has $\mathbf{y}_j$ as its *unique* nearest neighbor among the set of code vectors, then it must be assigned to R_j. If an input vector $\mathbf{x}$ is equally distant from two or more code vectors, the assignment of an index is not unique and we adopt the convention that it should always be assigned to the code vector with the smallest index among all the nearest neighbor code vectors. Any input point which does not have a unique nearest neighbor can be arbitrarily assigned to any of its nearest neighboring code vectors without altering the average distortion of the resulting quantizer.

Proof (General Case): For a given codebook, $\mathcal{C}$, the average distortion given by (11.2.1) can be lower bounded according to

$$D = \int d(\mathbf{x}, Q(\mathbf{x})) f_{\mathbf{X}}(\mathbf{x})\, d\mathbf{x} \geq \int [\min_{i \in \mathcal{I}} d(\mathbf{x}, \mathbf{y}_i)] f_{\mathbf{X}}(\mathbf{x})\, d\mathbf{x}, \tag{11.2.5}$$

where $\mathcal{I}$ is the index set $\{1, 2, \cdots, N\}$. This lower bound is attained if $Q(\cdot)$ assigns each $\mathbf{x}$ to a code vector that has the lowest distortion, in other words, if the nearest neighbor condition is satisfied.

We next consider the optimality of the codebook for a given partition. This leads to the *centroid condition* for specifying the code vector associated with each partition region. We define the *centroid*, cent(R), of any set $R \in \mathcal{R}^k$ with nonzero probability as that vector $\mathbf{y}$ (if it exists) which minimizes the distortion between a point $\mathbf{X}$ in R and $\mathbf{y}$ averaged over the probability distribution of $\mathbf{X}$ given that $\mathbf{X}$ lies in R. Thus,

$$\mathbf{y}^* = \text{cent}(R) \text{ if } E[d(\mathbf{X}, \mathbf{y}^*)|\mathbf{X} \in R] \leq E[d(\mathbf{X}, \mathbf{y})|\mathbf{X} \in R] \tag{11.2.6}$$

for all $\mathbf{y} \in \mathcal{R}^k$. It is convenient to write this as an *inverse minimum*, that is, (11.2.6) is equivalent to

$$\text{cent}(R) = \min_{\mathbf{y}}{}^{-1} E[d(\mathbf{X}, \mathbf{y})|\mathbf{X} \in R]$$

The notation *argmin* is also used sometimes instead of $\min^{-1}$. Thus the centroid is in some sense a natural representative or "central" vector for the set R and the associated probability distribution on R. It should be noted at this point that for some distortion measures, the centroid is not always uniquely defined. If the set R has zero probability, then the centroid can be defined in an arbitrary fashion.

For the squared error distortion measure, the centroid of a set R is simply the minimum mean squared estimate of $\mathbf{X}$ given that $\mathbf{X} \in R$. Similar to the proof of Theorem 4.2.1 this is easily seen to be

$$\text{cent}(R) = E(\mathbf{X}|\mathbf{X} \in R), \tag{11.2.7}$$

so that the general definition of the centroid reduces to the usual use of the term "centroid" as in mechanics and, in this case, the centroid is uniquely defined.

For the case of a finite set $R \in \mathcal{R}^k$, the centroid definition (11.2.7) remains applicable and can be evaluated if we know the pmf (probability mass function) for the input distribution. In particular, we shall be interested later in the case where each point in R has equal probability. For example, suppose the probability distribution is a sample distribution based on a *training set* of data. A training set is a finite collection of sample vectors generated from the source distribution in order to represent the statistics of the source with a finite set of data. When there is a natural ordering to the data, the training set is also called the *training sequence*. In the statistical literature training sets are also called *learning sets*. Usually, the training vectors are generated independently from the source. This gives rise to a

discrete model for the source where each of the L vectors in the training set has a probability of $1/L$. For the squared error measure the centroid then reduces to the arithmetic average

$$\text{cent}(R) = \frac{1}{||R||} \sum_{i=1}^{||R||} \mathbf{x}_i \tag{11.2.8}$$

for $R = \{\mathbf{x}_i;\ i = 1, 2, 3, \cdots, ||R||\}$, where $||R||$ is the *cardinality* of the set R, that is, the number of elements in R.

With this background, we state the condition for codebook optimality when the partition is given.

Centroid Condition

For a given partition $\{R_i; i = 1, \cdots, N\}$, the optimal code vectors satisfy

$$\mathbf{y}_i = \text{cent}(R_i). \tag{11.2.9}$$

The result is an exact generalization of the centroid condition for scalar quantizers of Chapter 6. It is proved in three different ways below. Each proof gives a somewhat different insight.

First Proof (General Case): The average distortion is given by

$$D = \sum_{i=1}^{N} \int_{R_i} d(\mathbf{x}, \mathbf{y}_i) f_{\mathbf{X}}(\mathbf{x})\, d\mathbf{x} = \sum_{i=1}^{N} P_i \int d(\mathbf{X}, \mathbf{y}_i) f_{\mathbf{X}|i}(\mathbf{x})\, d\mathbf{x},$$

where we use the notation $f_{\mathbf{X}|i}(\mathbf{x})$ to mean the conditional pdf for $\mathbf{X}$ given $\mathbf{X} \in R_i$ and where $P_i = P[\mathbf{X} \in R_i]$. Since the partition is fixed, each term can be separately minimized by finding $\mathbf{y}_i$ that will minimize the expected distortion, or

$$E[d(\mathbf{x}, \mathbf{y}_i)|\mathbf{X} \in R_i] = \int d(\mathbf{X}, \mathbf{y}_i) f_{\mathbf{X}|i}(\mathbf{x})\, d\mathbf{x}. \tag{11.2.10}$$

By definition, the centroid $\mathbf{y}_i = \text{cent}(R_i)$ minimizes this conditional distortion. □

The following two proofs are restricted to the case where the squared error distortion measure is used as the performance measure. They are based on the view that the decoder receives partial information about the input vector and must perform a statistical estimate of the input given the transmitted index. The second proof finds the optimal decoder by finding the optimal (not necessarily linear) estimator of X and the third proof is based on the recognition that the decoder always performs a linear operation and the optimal estimate is linear.

As background for these proofs, we introduce the binary vector $\mathbf{S}$ of selector functions following the treatment in [138] for modeling vector quantizers and defined in a manner similar to the presentation of scalar quantization in Chapter 5. Since the partition is fixed, the selector functions, $S_i(\mathbf{x}) = 1_{R_i}(\mathbf{x})$, defined in Chapter 10, are well-defined and the encoder output can be expressed in terms of the binary vector

$$\mathbf{S} = (S_1(\mathbf{X}), \cdots, S_N(\mathbf{X})) \tag{11.2.11}$$

where for each $i \in \mathcal{I}$, $S_i = S_i(\mathbf{X})$ has value unity when $\mathbf{X} \in R_i$ and value zero otherwise. Then the N-dimensional vector $\mathbf{S} = (S_1, S_2, \cdots, S_N)$ fully describes the output of the encoder. This vector can only take on values in which all components but one are zero and the remaining component has value unity. With this background, the remaining proofs can be presented.

Second Proof (Squared Error Distortion): The decoder can be regarded as a vector-valued function of $\mathbf{S}$, with output $\mathbf{Y} = \mathbf{F}(\mathbf{S})$ where $\mathbf{Y}$ is a k-dimensional reproduction vector. To minimize $D = E[||\mathbf{X} - \mathbf{F}(\mathbf{S})||^2]$, we have from Theorem 4.2.1 that $F(\mathbf{S}) = E[\mathbf{X}|\mathbf{S}]$ or, in terms of sample values,

$$\mathbf{F}(\mathbf{z}) = E[\mathbf{X}|\mathbf{S} = \mathbf{z}]. \tag{11.2.12}$$

Now, recognizing that only one component of the binary vector $\mathbf{z}$ can be nonzero, there are only N distinct values that can be taken on by $\mathbf{z}$. In particular, suppose $\mathbf{z} = \mathbf{u}_i$, where $\mathbf{u}_i$ has a 1 in the ith coordinate and 0 elsewhere. Then

$$\mathbf{F}(\mathbf{u}_i) = \mathbf{y}_i = E[\mathbf{X}|S_i = 1],$$

which again proves the necessity of the centroid condition for optimal quantization. □

Third Proof (Squared Error Distortion): From the primary decomposition of a vector quantizer, we can express the reproduction vector, $\mathbf{Y}$, as

$$\mathbf{Y} = Q(\mathbf{X}) = \sum_{j=1}^{N} \mathbf{y}_j S_j. \tag{11.2.13}$$

It may be seen that this is a *linear* combination of the observable random variables S_j. Since we wish to minimize the average distortion $E(||\mathbf{X} - \mathbf{Y}||^2)$, we seek the best estimate of $\mathbf{X}$ using the estimator $\mathbf{Y}$, which is constrained to be a linear combination of well-defined random variables, S_j. By the orthogonality principle (Theorem 4.2.4), the condition for optimality is that the estimation error $\mathbf{X} - \mathbf{Y}$ must be orthogonal to each of the observable variables, S_i. Thus we have

$$E(\mathbf{X}S_i) = \sum_{j=1}^{N} \mathbf{y}_j E(S_j S_i). \tag{11.2.14}$$

Note that $E(S_j S_i)$ is zero for $j \neq i$ and is equal to $ES_i^2 = P_i$ when $j = i$. Also, the expectation $E(\mathbf{X}S_i)$ can be expressed in the form

$$E(\mathbf{X}S_i) = E[\mathbf{X}|S_i = 1]P_i. \tag{11.2.15}$$

From these observations we see that (11.2.14) simplifies and yields:

$$\mathbf{y}_i = \frac{E(\mathbf{X}S_i)}{ES_i^2} = E[\mathbf{X}|S_i = 1] = \text{cent}(R_i). \tag{11.2.16}$$

□

This result assumes that the partition is *nondegenerate* in the sense that each region has nonzero probability of containing the input vector, i.e., $P_i \neq 0$. In the degenerate case where $P_i = 0$, we have what is called the *empty cell problem.* For such a partition region, the centroid is undefined and clearly it makes no sense to have a code vector dedicated to representing this cell. This situation is not of theoretical concern since a quantizer with a code vector representing an empty cell will not be optimal: the cell can be omitted from the partition and another cell with positive probability can be split into two cells thereby keeping the codebook size unchanged while reducing the average distortion. In practice however, the empty cell problem must be considered in codebook design as will be discussed later in this chapter.

It is noteworthy that the decoder of any vector quantizer always performs a *linear estimate* of the input vector based on partial information about that input as given by the binary observables S_i. The optimal decoder for a given encoder is therefore always an *optimal linear estimate* of the input vector. The restriction to the squared error distortion was needed only to obtain a simple and explicit solution for the optimal linear estimator (or equivalently, for the optimal code vectors). For other distortion measures, the optimal linear estimator is computable, but may be less tractable analytically. Later we shall consider the solution for other important distortion measures.

The two necessary conditions are direct generalizations of the scalar case as derived by Lloyd and others. These conditions are so fundamental to quantization and more generally to source coding, that they recur in similar forms in many other contexts. It is essential that the reader clearly grasp the two necessary conditions for optimality.

We shall see in the next section that these two necessary conditions are not sufficient for optimality. It is hence of interest to know if there are additional necessary conditions that might be used as further tests for optimality. There is a third condition, due also to Lloyd in the scalar case, which is useful in the case of discrete alphabets.

Given a vector quantizer satisfying the nearest neighbor condition and the centroid condition, let $\{\mathbf{y}_i\}$ denote the reproduction points, $\{R_j\}$ the partition cells, and $\{R'_j\}$ the sets

$$R'_j = \{\mathbf{x} : d(\mathbf{x}, \mathbf{y}_j) \leq d(\mathbf{x}, \mathbf{y}_i), \text{ for all } i \in \mathcal{I}\}.$$

Thus R_j consists of the nearest neighbor cell R_j augmented by its boundary points. From the NN Condition $R_j \subset R'_j$. Call the set $B_j = R'_j - R_j$ of points in R'_j but not in R_j the *boundary* of R_j. The boundary consists of points which are not in R_j, but which are equally close to both $\mathbf{y}_j$ and to some other $\mathbf{y}_i$ and hence do not have a unique nearest neighbor. This leads to the following necessary condition for optimality.

Zero Probability Boundary Condition

A necessary condition for a codebook to be optimal for a given source distribution is

$$P(\bigcup_{j=1}^{N} B_j) = 0,$$

that is, the boundary points occur with zero probability. An alternative way to give this condition is to require that the collection of points equidistant from at least two code words has probability 0, that is,

$$P(\mathbf{x} : d(\mathbf{x}, \mathbf{y}_i) = d(\mathbf{x}, \mathbf{y}_j) \text{ for some } i \neq j) = 0.$$

Proof: Suppose that $P(B_j) \neq 0$ for some j and hence there is at least one input point, say x_0, which is encoded into $\mathbf{y}_j$ but which is equally distant from another code vector y_i. Thus a new partition formed by breaking the tie in a different way, that is, mapping x_0 into the other reproduction value y_i, will yield a code with the same average distortion. This results, however, in moving a nonzero probability input point into the cell of a different reproduction vector, $\mathbf{y}_i$, which must move the centroids of both R_j and R_i, which means that the codebook is no longer optimal for the new partition. □

The zero-probability boundary condition is useless (i.e., automatically satisfied) when the input is a continuous random variable since for all the distortion measures that we have considered, the boundary will have zero volume and hence zero probability. The condition can, however, be useful in the discrete case since then probability can easily be placed on boundary points, e.g., one of the training samples may be equally distant from two reproduction vectors. The above condition states that the resulting quantizer cannot be optimal even if the nearest neighbor and centroid conditions are satisfied. This suggests that a better quantizer can be found by breaking the tie in a different manner and proceeding with more iterations.

Sufficiency of the Optimality Conditions

Suppose one has a vector quantizer that satisfies the centroid condition, the nearest neighbor condition, and the zero-probability boundary condition. Does this guarantee that it is globally or even locally optimal? First, we must be more explicit about the meaning of "global" and "local" optimality. We are trying to minimize the average distortion, D, which is a function of both the partition and the codebook. Since the NN condition is necessary for optimality, let us assume that it is satisfied for any given codebook $\mathcal{C}$. Then we may regard the performance measure D as an explicit function of the codebook; the distortion is a multivariate function of each component of each vector. Thus the average distortion is simply a function of kN real variables. Now, the idea of optimality is quite simple. The quantizer is *locally optimal* if every small perturbation of the code vectors does not lead to a decrease in D. (Any such perturbation implies the nearest neighbor regions are correspondingly perturbed.) It is *globally optimal* if there exists no other codebook that gives a lower value of D. If we have a codebook that satisfies both necessary conditions of optimality, it is widely believed that it is indeed locally optimal. No general theoretical derivation of this result has ever been obtained. For the particular case of a discrete input distribution such as a sample distribution produced by a training sequence, however, it can be shown that under mild restrictions, a vector quantizer satisfying the necessary conditions is indeed locally optimal. (See Appendix C of [167].) This is intuitively evident since in the discrete input case, a slight perturbation of a code vector will not alter the partitioning of the (countable) set of input vectors as long as none of the input values lies on a partition boundary; once the partition stays fixed, the perturbation causes the centroid condition to be violated and the average distortion can only increase. At least under these conditions, a vector quantizer that satisfies the necessary conditions will be locally optimal.

The designer should be aware that locally optimal quantizers can, in fact, be very suboptimal. Examples of such quantizers may be found in [166], but one example is fairly easy to visualize. Suppose one has an optimal 3 bit scalar quantizer for the unit interval. One can design a two-dimensional quantizer for the unit square, also having 3 bits, by using the scalar quantizer on one coordinate and ignoring the other. This produces a partition of the plane into eight parallel stripes whose widths correspond to the quantizing intervals of the optimal scalar quantizer. The resulting quantizer is clearly poor since it completely ignores one coordinate, yet it satisfies the Lloyd conditions.

Implications of Optimal Vector Quantizers

The following two properties follow from the optimality conditions for the squared error distortion measure.

Lemma 11.2.1 *A vector quantizer which satisfies the necessary conditions for optimality with the squared error distortion measure is regular.*

Proof: We have seen in Chapter 10 that the partition cells are convex for vector quantizers satisfying the NN condition with the squared error distortion measure. If the pdf of a random vector is distributed over a convex set in $\mathcal{R}^k$, its mean must lie within the set. (See Problem 11.10.) Hence the centroid condition implies that each code vector lies within its corresponding partition cell. Therefore, the quantizer is regular. □

Lemma 11.2.1 can readily be generalized to the case of any distortion measure that is a monotonic function of Euclidean distance. (See Problem 11.8.) The next lemma generalizes the results of Lemma 6.2.2 to vectors.

Lemma 11.2.2 *A vector quantizer which satisfies the centroid condition for the squared error distortion measure has*

(a) $EQ(\mathbf{X}) = E\mathbf{X}$,

(b) $E(\mathbf{X}^t Q(\mathbf{X})) = E(||Q(\mathbf{X})||^2)$, *and*

(c) $E(||Q(\mathbf{X})||^2) = E(||\mathbf{X}||^2) - E(||\mathbf{X} - Q(\mathbf{X})||^2)$.

Proof: Since the decoder is optimal for the given encoder, the orthogonality condition is satisfied, i.e., $E[(\mathbf{X} - Q(\mathbf{X}))S_i] = 0$ for each i. Summing these equations over i gives $E[(\mathbf{X} - Q(\mathbf{X}))(\sum_i S_i)] = 0$. But the sum $\sum_i S_i$ is always unity due to the special character of the variables S_i defined earlier. Hence, $E(\mathbf{X} - Q(\mathbf{X})) = 0$ and part (a) is proved. Now, recalling that Y is a linear combination of the S_i's, the orthogonality condition implies that

$$E[(\mathbf{X} - Q(\mathbf{X}))^t Q(\mathbf{X})] = 0,$$

showing that the quantization error is uncorrelated with the output vector. This result immediately implies part (b), that $E(\mathbf{X}^t Q(\mathbf{X})) = E(||Q(\mathbf{X})||^2)$. Also, it follows that

$$D \equiv E(||\mathbf{X} - Q(\mathbf{X})||^2) = E[(\mathbf{X} - Q(\mathbf{X}))^t \mathbf{X}].$$

Hence

$$\begin{aligned} E(||\mathbf{X} - Q(\mathbf{X})||^2) &= E(||\mathbf{X}||^2) - E[\mathbf{X}^t Q(\mathbf{X})] \\ &= E(||\mathbf{X}||^2) - E(||Q(\mathbf{X})||^2). \end{aligned}$$

□

Lemma 11.2.2(a) demonstrates that the study of optimal quantization can focus on zero mean random input vectors without loss of generality. Given the optimal quantizer for a zero mean random vector, assume we wish to modify it for input vectors having a constant mean μ added to them. We simply add μ to each code vector and the resulting codebook is again optimal.

Lemma 11.2.2(b) shows that the quantization error $Q(\mathbf{X}) - \mathbf{X}$ is *always* correlated with the input $\mathbf{X}$ if the vector quantizer has an optimal decoder for its encoder. This implies that a model of vector quantization as the addition of an independent "noise" vector to the input vector cannot be valid or at least cannot be strictly correct. In high resolution VQ, however, it may offer a useful approximation (see, e.g., [159], Chapter 5).

Since the quantization error energy can never be zero for a continuously distributed input vector, Lemma 11.2.2(c) shows that the quantized vector $Q(\mathbf{X})$ always has less energy than the input vector $\mathbf{X}$. This statement also implies that the additive independent noise vector model is not valid, since such a model would imply that $Q(\mathbf{X})$ has larger energy than $\mathbf{X}$. At high resolution, the average distortion is very small so that the input and output energies are almost equal.

11.3 Vector Quantizer Design

The necessary conditions for optimality provide the basis for iteratively improving a given vector quantizer. If the iteration continues to convergence, a good (hopefully close to optimal) quantizer can be found. The iteration begins with a vector quantizer consisting of its codebook and the corresponding optimal (NN) partition and then finds the new codebook which is optimal for that partition. This new codebook and its NN partition are then a new vector quantizer with average distortion no greater (and usually less) than the original quantizer. Although each of these steps of optimizing a partition for a codebook and a codebook for a partition is simple and straightforward, the simultaneous satisfaction of both conditions is not easy to achieve. There are no known closed-form solutions to the problem of optimal quantization. The repeated application of the improvement step, however, yields an *iterative* algorithm which at least reduces (or leaves unchanged) the average distortion at each step. In effect, we design a sequence of vector quantizers, which continually improve in performance if the algorithm is effective.

We begin with the problem of obtaining the initial codebook for improvement since this, too, is a problem of vector quantizer design. In fact,

if the initial codebook is good enough, it may not be worth the effort to run further improvement algorithms. There are a variety of techniques for generating a codebook that have been developed in cluster analysis (for pattern recognition) and in vector quantization (for signal compression). We survey several of the most useful.

Random Coding

Perhaps the simplest conceptual approach towards filling a codebook of N code words is to randomly select the code words according to the source distribution, which can be viewed as a Monte Carlo codebook design. The obvious variation when designing based on a training sequence is to simply select the first N training vectors as code words. If the data is highly correlated, it will likely produce a better codebook if one takes, say, every Kth training vector. This technique has often been used in the pattern recognition literature and was used in the original development of the k-means technique [227]. One can be somewhat more sophisticated and randomly generate a codebook using not the input distribution, but the distribution which solves the optimization problem defining Shannon's distortion-rate function. In fact, the Shannon source coding theorems imply that such a random selection will on the average yield a good code [134][32][159]. Unfortunately, the codebook will have no useful structure and may turn out quite awful.

Observe that here "random coding" means only that the codebook is selected at random: once selected it is used in the usual deterministic (nearest neighbor) fashion. This is the same sense that "random coding" is used in information theory.

Pruning

Pruning refers to the idea of starting with the training set and selectively eliminating (pruning) training vectors as candidate code vectors until a final set of training vectors remains as the codebook. In once such method, a sequence of training vectors is used to populate a codebook recursively as follows: put the first training vector in the codebook. Then compute the distortion between the next training vector and the first code word. If it is less than some threshold, continue. If it is greater than the threshold, add the new vector to the codebook as a codeword. With each new training vector, find the nearest neighbor in the codebook. If the resulting distortion is not within some threshold, add the training vector to the codebook. Continue in this fashion until the codebook has enough words. For a given finite set of training vectors, it is possible that the resulting codebook will

have fewer than the desired number of code vectors. If this happens, the threshold value must be reduced and the process repeated. A typical choice of threshold value for the MSE distortion measure is proportional to $N^{-\frac{2}{k}} = 2^{2r}$ where r is the rate of the code. (See Problem 11.11.) This technique is well known in the statistical clustering literature. (See, e.g., Tou and Gonzales [308].)

Pairwise Nearest Neighbor Design

A more complicated, but better, means of finding a codebook from a training sequence is the pairwise nearest neighbor (PNN) clustering algorithm proposed by Equitz [106][107]. Similar algorithms have also been used in the clustering literature [326][9]. This is also a form of pruning as it begins with the entire training sequence of L vectors, and ends with a collection of N vectors. Unlike the previous design technique, however, the final vectors need not be in the training sequence. The technique involves more computation than the preceding methods, but it is faster than the generalized Lloyd algorithm which attempts to optimize the codebook.

Suppose that the training sequence has L vectors, each of which is considered to be a separate cluster containing a single vector. The goal will be to merge vectors together into groups or clusters until we have the desired number, say N. The codebook will then contain the centroids of these clusters. In this manner we will have a partition of the training sequence into the correct number of cells and have the optimal codebook for this partition. The partition might not be a nearest neighbor partition, however. The induced vector quantizer would then replace this partition by the NN partition. The partition is obtained as follows. First compute the distortion between all pairs of vectors. The two training vectors having the smallest distortion are combined into a single cluster and represented by their centroid. We now have $L-1$ clusters, one containing two vectors and the rest containing a single vector. Henceforth at each step clusters may have more than one vector. Suppose now that we have K clusters with $N < K \leq L-1$ and we wish to merge two of the clusters to get a good set of $K-1$ clusters. This single step merging can be done in an optimal fashion as follows: For every pair of clusters drawn from the full collection of K clusters, compute the increase in average distortion resulting if the two clusters and their centroids are replaced by the merged two clusters and the corresponding centroid. This computation is much easier than it sounds in the case of squared error. When the best pair of clusters for merging is found, they are merged to form a codebook with $K-1$ vectors. Continue in this way until only N vectors remain. Thus, for example, each of the K^2 pairs of clusters consists of two clusters of vectors $R_i = \{x_i(l);\ l = 1, 2, \cdots, L_i\}$ and

$R_j = \{x_j(l);\ l = 1, 2, \cdots, L_j\}$. The contribution of these two clusters to the average distortion if they are not merged is

$$\Delta_{i,j} = \sum_{l=1}^{L_i} d(x_i(l), \text{cent}(R_i)) + \sum_{l=1}^{L_j} d(x_j(l), \text{cent}(R_j))$$

while the contribution if they are merged is

$$\Delta'_{i,j} = \sum_{l=1}^{L_i} d(x_i(l), \text{cent}(R_i \bigcup R_j)) + \sum_{l=1}^{L_j} d(x_j(l), \text{cent}(R_i \bigcup R_j)) \geq \Delta_{i,j}.$$

The pair of clusters R_i, R_j for which $\Delta'_{i,j} - \Delta_{i,j}$ is the smallest is merged. That is, the two clusters are merged which thereby cause the least increase in the overall average distortion.

Note that each merge is optimal, but the overall procedure need not be optimal, that is, need not produce the optimal codebook of the given size.

Product Codes

In some cases a product codebook may provide a good initial guess. For example, if one wishes to design a codebook for a k-dimensional VQ with codebook size 2^{kR} for some integral resolution R, then one can use the product of k scalar quantizers with 2^R words each. Thus if $q(x)$ is a scalar quantizer, then $Q(x_0, \cdots, x_{k-1}) = (q(x_0), \cdots, q(x_{k-1}))$, the Cartesian product of the scalar quantizers, is a vector quantizer. This technique will not work if R is not an integer. In general other product structures can be used, e.g., one could first design a one-dimensional quantizer q_1 from scratch (perhaps using a uniform quantizer as an initial guess). One could then use $(q_1(x_0), q_1(x_1))$ as an initial guess to design a good two-dimensional quantizer $q_2(x_0, x_1)$. One could then initiate a three-dimensional VQ design with the product $(q_1(x_0), q_2(x_1, x_2))$ as an initial guess. One could continue in this way to construct higher dimensional quantizers until the final size is reached.

Splitting

Linde et al. introduced a technique that resembles the product code initialization in that it grows large codebooks from small ones, but differs in that it does not require an integral number of bits per symbol [215]. The method is called the *splitting algorithm* and it produces increasingly larger codebooks of a fixed dimension. The globally optimal resolution 0 codebook of a training sequence is the centroid of the entire sequence. The one

code word, say $\mathbf{y}_0$, in this codebook can be "split" into two code words, $\mathbf{y}_0$ and $\mathbf{y}_0 + \epsilon$, where ϵ is a vector of small Euclidean norm. One choice of ϵ is to make it proportional to the vector whose ith component is the standard deviation of the ith component of the set of training vectors. Another choice is to make it proportional to the eigenvector corresponding to the largest eigenvalue of the covariance matrix of the training set. This new codebook has two words and can be no worse than the previous codebook since it contains the previous codebook. The iterative improvement algorithm can be run on this codebook to produce a good resolution 1 code. When complete, all of the code words in the new codebook can be split, forming an initial guess for a resolution 2 codebook. One continues in this manner, using a good resolution r codebook to form an initial resolution $r + 1$ codebook by splitting. This algorithm provides a complete design technique from scratch on a training sequence and will later be seen to suggest a vector analog to successive approximation quantizers.

The Generalized Lloyd Algorithm

We now return to the iterative codebook improvement algorithm based on the necessary conditions for optimality, the generalization of Lloyd's Method I for designing scalar quantizers. We have just surveyed a collection of techniques which produce VQ codebooks, but none of them will in general produce a codebook meeting Lloyd's necessary conditions for optimality. Any of the codebooks can, however, be used as an initial codebook for application of the Lloyd iterative improvement algorithm. We now delve more deeply into this algorithm for designing vector quantizers.

The generalized Lloyd algorithm for VQ design is sometimes known as the *k-means algorithm* after MacQueen [227] who studied it as a statistical clustering procedure. The idea had been described earlier by Forgey in 1965 [122] in a clustering context. It has since led to a variety of extensions and applications in in the statistical literature. (See, e.g., Diday and Simon [101].) It is also sometimes referred to as the LBG algorithm in the data compression literature after [215] where a detailed treatment of the algorithm for data compression applications is presented, although this name is perhaps more appropriate for the splitting algorithm variation of the Lloyd algorithm. We shall refer to it as the generalized Lloyd or GL algorithm (or the GLA) since it is indeed a direct generalization of Lloyd's treatment in 1957 [218]. For design based on empirical data, it was first used for VQ in 1977 by Chen for the design of two-dimensional quantizers [60] and by Hilbert [178] for the design of multispectral image compression systems.

The algorithm is based on the iterative use of the codebook modification operation, which generalizes the Lloyd Iteration for scalar quantization. We

first state the generalization for the vector case when the joint pdf of the input vector is assumed to be known and continuously distributed.

The Lloyd Iteration for Codebook Improvement
Known Statistics:

(a) Given a codebook, $\mathcal{C}_m = \{\mathbf{y}_i;\ i = 1, \cdots, N\}$, find the optimal partition into quantization cells, that is, use the Nearest Neighbor Condition to form the nearest neighbor cells:

$$R_i = \{\mathbf{x} : d(\mathbf{x}, \mathbf{y}_i) < d(\mathbf{x}, \mathbf{y}_j);\ \text{all } j \neq i\}.$$

If $\mathbf{x}$ yields a tie for distortion, e.g., if $d(\mathbf{x}, \mathbf{y}_i) = d(\mathbf{x}, \mathbf{y}_j)$ for one or more $j \neq i$, then assign $\mathbf{x}$ to the set R_j for which j is smallest.

(b) Using the Centroid Condition, find $\mathcal{C}_{m+1} = \{\text{cent}(R_i);\ i = 1, \cdots, N\}$, the optimal reproduction alphabet (codebook) for the cells just found.

Lemma 11.3.1 *Each application of the Lloyd iteration must reduce or leave unchanged the average distortion.*

Proof: From the necessary conditions for optimality it follows that Step (a) in the algorithm can only improve or leave unchanged the encoder for the given decoder. Similarly Step (b) can only improve or leave unchanged the decoder for the given encoder. Hence, the average distortion of the vector quantizer cannot increase. □

Note that the lemma does not actually prove that the sequence of codebooks actually converges to a local optimum, nevertheless practical implementations of the algorithm have been found to be very effective. The above form of the Lloyd Iteration requires that the input pdf be known and that the geometry of the partition be specified in order to compute the centroids. If the input pdf happens to have regions of zero probability it is possible that for some choices of codebook $\mathcal{C}_m$ the empty cell problem can arise and one or more centroids cannot be computed in Step (b). The computation of the centroids in Step (b), by evaluating a multiple integral over a complicated region in $\mathcal{R}^k$, is generally impossible by analytical methods even if the pdf has a simple and tractable expression. In practice, a

numerical integration would be essential to find the centroids. However, an adequate analytical description of the input pdf is generally not available in most applications. Instead, a sample distribution based on empirical observations of the input vector is used to generate the improved codebook. This approach has become a standard technique for quantizer design in recent years. In fact, this approach to be described next is equivalent to a Monte Carlo method for evaluating the needed integrals that determine the centroids.

Suppose we have a set of observations of the signal to be quantized, called the *training set*,

$$\mathcal{T} = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_M\}, \tag{11.3.1}$$

where M is the size of the training set. We define the *training ratio* as $\beta \equiv M/N$, where N is the codebook size. This parameter indicates how effectively the training set can describe the true pdf of the source and how well the Lloyd algorithm can yield a codebook that is nearly optimal for the true pdf. The training set can be used to define statistically a random vector $\mathbf{U}$ by assigning the probability mass function (pmf) to have mass $1/M$ for each value of $\mathbf{v}_i$ in $\mathcal{T}$. Although a pdf does not exist for a discrete distribution (unless we use delta functions), the cumulative distribution function is well defined for $\mathbf{U}$. Specifically, the *empirical cdf* of $\mathbf{U}$ is given by

$$F^{(M)}(\mathbf{x}) = \frac{1}{M} \sum_{i=1}^{M} \hat{u}(\mathbf{x} - \mathbf{v}_i) \tag{11.3.2}$$

where $\hat{u}(\mathbf{w})$ denotes the unit vector-step function which is one if *all* components of $\mathbf{w}$ are nonnegative and zero otherwise.

The strong law of large numbers or the pointwise ergodic theorem implies that the empirical cdf converges with probability one to the actual cdf of the input random variable (at all points where the cdf is continuous) as the training size M approaches infinity. (See, e.g., [251] for a discussion of the convergence of empirical distributions for general asymptotically stationary processes.) If we find an N point vector quantizer that is optimal for input vectors whose distribution is given by the empirical cdf of the training set and if the training set is large enough, we can expect the quantizer to be very nearly optimal for the actual input vector (which generally has a continuous cdf). Conditions under which such convergence can be mathematically guaranteed are developed in [167] [279].

For a discrete input, the selector functions $S_i(\mathbf{x})$ (defined in Chapters 5 and 10) are now defined for a given partition of $\mathcal{T}$ as $S_i(\mathbf{x}) = 1$ if $\mathbf{x} \in R_i$.

Hence the Centroid Condition for a cell R_i is given by

$$\mathbf{Y}_j = E[\mathbf{X}|\mathbf{X} \in R_j] = \frac{E[\mathbf{X}S_j(\mathbf{X})]}{E[S_j(\mathbf{X})]}. \tag{11.3.3}$$

Thus we have

$$\mathbf{Y}_j = \frac{\frac{1}{M}\sum_{i=1}^{M} \mathbf{v}_i S_j(\mathbf{v}_i)}{\frac{1}{M}\sum_{i=1}^{M} S_j(\mathbf{v}_i)} \tag{11.3.4}$$

for $j = 1, 2, \cdots, N$. The index i counts training points and the index j counts partition cells. Again, we note that the empty cell problem will arise if a region R_j has zero probability, i.e., if there are no training points that lie in this set. In this case, the centroid (11.3.4) is undefined.

The nearest neighbor condition for the discrete input case give the partition regions:

$$R_j = \{\mathbf{v} \in \mathcal{T} : ||\mathbf{v} - \mathbf{y}_j|| < ||\mathbf{v} - \mathbf{y}_i|| \text{ all } i\}. \tag{11.3.5}$$

For the discrete case, the average distortion can be expressed as

$$D = \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{N} d(\mathbf{v}_i, \mathbf{y}_j) S_j(\mathbf{v}_i) \tag{11.3.6}$$

where the quantizer input is confined to values in the finite set $\mathcal{T}$, and $\mathcal{C} = \{\mathbf{y}_j\}$ is the codebook.

The Lloyd iteration can now be directly applied to the discrete input distribution defined from the training set $\mathcal{T}$ to obtain a locally optimal quantizer for this distribution. The general conclusion here is that if we design an optimal N point quantizer for a finite training ratio, it is reasonable to expect that for M sufficiently large, it will be very nearly optimal for the true distribution of the input random vector.

The Lloyd Iteration for Empirical Data

(a) Given a codebook, $\mathcal{C}_m = \{\mathbf{y}_i\}$, partition the training set into cluster sets R_i using the Nearest Neighbor Condition:

$$R_i = \{\mathbf{x} \in \mathcal{T} : d(\mathbf{x}, \mathbf{y}_i) \leq d(\mathbf{x}, \mathbf{y}_j);\ \text{all } j \neq i\}$$

(and a suitable tie-breaking rule).

(b) Using the Centroid Condition, compute the centroids for the cluster sets just found to obtain the new codebook, $\mathcal{C}_{m+1} = \{\text{cent}(R_i)\}$. If an empty cell was generated in Step (a), an alternate code vector assignment is made (in place of the centroid computation) for that cell.

A variety of heuristic solutions have been proposed for handling the empty cell problem. In some programs a cell is declared "empty" if it has 3 or fewer training vectors. In one method, the cell with the highest number of training vectors is assigned a second code vector by splitting its centroid into two vectors and the empty cell is deleted. Some researchers use the cell with the highest partial distortion as the one whose centroid is to be split. If c cells are empty in a given iteration, the c cells with highest partial distortion can have their centroids split.

Now that we have defined the codebook improvement iteration, the actual design algorithm is stated concisely in Table 11.1 Note that Step 2 should include a suitable heuristic for handling the empty cell problem. Although not usually included in most implementations of the GL algorithm, one could check before terminating if the zero-probability boundary condition is satisfied, i.e., that no training vector is equidistant from two (or more) code vectors. If this is not satisfied and the algorithm terminates then the resulting codebook can in principle be improved by re-assigning such a training vector to a different cell (i.e., one not consistent with the tie-breaking rule) and then performing an additional Lloyd iteration. The check for ties is easily done as part of Step (a) of the Lloyd Iteration. In practice, this additional step will ordinarily offer only a negligible improvement when a reasonable large training ratio is used. For very low training ratios this step can be an important one.

The flow chart for the generalized Lloyd algorithm is shown in Figure 11.3. Although various stopping criteria can be used, it is common and effective to test if the fractional drop in distortion, $(D_m - D_{m+1})/D_m$,

Table 11.1: **The Generalized Lloyd Algorithm**

Step 1. Begin with an initial codebook $\mathcal{C}_1$. Set $m = 1$.

Step 2. Given the codebook, $\mathcal{C}_m$, perform the Lloyd Iteration to generate the improved codebook $\mathcal{C}_{m+1}$.

Step 3. Compute the average distortion for $\mathcal{C}_{m+1}$. If it has changed by a small enough amount since the last iteration, stop. Otherwise set $m + 1 \rightarrow m$ and go to Step 2.

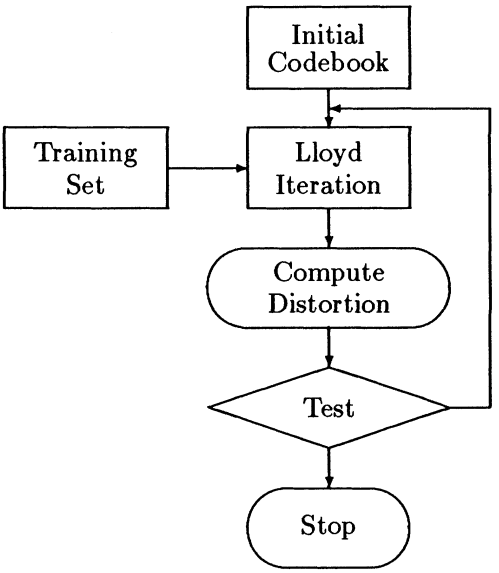


Figure 11.3: Lloyd Algorithm Flow Chart

is below a suitable threshold. When this happens and the zero-probability boundary condition is satisfied, then the algorithm is halted. With a threshold value of zero, the algorithm will necessarily produce a sequence of codebooks with monotone nonincreasing values of average distortion. If the algorithm converges to a codebook in the sense that further iterations no longer produce any changes in the reproduction values, then the resulting codebook must simultaneously satisfy both necessary conditions for optimality.

Lemma 11.3.2 *For a finite training set, the GL algorithm always produces a sequence of vector quantizers whose average distortions converge in a finite number of iterations.*

Proof: There is a finite number of ways to partition the training set into N subsets. For each partition, the centroid condition yields a codebook with a particular value of average distortion. The monotone nonincreasing average distortion implies that the algorithm cannot return in subsequent iterations to a partition yielding a higher average distortion. Hence, the average distortion of the sequence of vector quantizers produced must converge to a fixed value in a finite number of steps. □

The GL algorithm can be initialized by selecting a codebook that has code vectors reasonably well scattered over the space and somehow covers the principal region of interest, where the input pdf is fairly large. All of the previously described VQ design techniques can be used to generate initial guesses, but among the most popular are the random code, splitting, and pairwise nearest neighbor techniques.

Although we have focused on the GL algorithm as the centerpiece of VQ design, a number of alternative approaches to codebook design exist. In this section we briefly sketch the main idea of some of these alternatives. The special attention to the GL algorithm is, however, justified, not only because of its prevalence in the engineering literature but also because it can always be used as an add-on to any other design technique. Since the GL algorithm can only improve (or at worst leave unchanged) the performance of any given initial codebook, any alternative VQ design method can always be regarded as a way to generate an initial codebook. Of course, any truly effective design algorithm ought to leave little or no room for improvement by the GL algorithm.

Before discussing specific methods, it is important to emphasize that the codebook design problem for a nearest neighbor quantizer is equivalent to the minimization of a multimodal performance measure over a high dimensional space. The GL algorithm is a *descent* algorithm, meaning that each iteration always reduces (or at least never increases) average distortion. Furthermore, each Lloyd iteration generally corresponds to a *local*

change in the codebook, that is, the new codebook is generally not drastically different from the old codebook. These properties suggest that once an initial codebook is chosen, the algorithm will lead to the nearest local minimum to the initial codebook in the space of all possible codebooks. Since a complex cost function will generally have many local minima and some can be much better than others, it is clear that the GL algorithm does not have the ability to locate an *optimal* codebook.

Stochastic Relaxation

By introducing randomness into each iteration of the GL algorithm it becomes possible to evade local minima, reduce or eliminate the dependence of the solution on the initial codebook, and locate a solution that may actually be a global minimum of the average distortion as a function of the codebook. A family of optimization techniques called *stochastic relaxation* (SR) is characterized by the common feature that each iteration of a search for the minimum of a cost function (e.g., the average distortion) consists of perturbing the *state*, the set of independent variables of the cost function (e.g., the codebook) in a random fashion. The magnitude of the perturbations generally decreases with time, so that convergence is achieved. A key feature of SR algorithms is that increases in the value of the cost function are possible in each iteration. An important family of SR algorithms known as *simulated annealing* (SA) are considered below and treated in [210].

The earliest use of randomness for VQ design was proposed in [215] where noise is added to the training vectors prior to a Lloyd iteration and the variance of the noise is gradually decreased to zero. This is indeed an SR algorithm although it was not so labelled. A convenient and effective SR algorithm for codebook design was introduced in [350] and is a simpler version of earlier methods proposed in [319] and [52]. This technique modifies the Lloyd iteration in the GL algorithm in the following very simple manner: After each centroid calculation, zero-mean noise is added to each component of each code vector. The variance of the noise is reduced with successive iterations according to a predetermined schedule. Corresponding to the simulated annealing literature, the terminology *temperature* is used to refer to the noise variance and a *cooling schedule* to refer to the sequence $\{T_m\}$ of temperature values as a function of the iteration number m. An effective cooling schedule was found to be

$$T_m = \sigma_x^2(1 - \frac{m}{I})^3,$$

where σ_x^2 is the average variance of the components of the source vector and I is the total number of iterations to be performed. The noise samples

were generated as independent uniform random variables (whose range is determined by the variance). In one example, an improvement of 0.8 dB in SNR was reported in [350] over the standard GL algorithm, although typically the improvements achieved are more modest. In some applications, even a gain of 0.5 dB in SNR can be of importance and since the codebook design is off-line, it may well be worth the increase in design computation to obtain even a slightly enhanced codebook. A more extensive discussion of SR algorithms for VQ design is given in [351].

Simulated Annealing

Simulated annealing is a stochastic relaxation technique in which a randomly generated perturbation to the state (the codebook in our context) at each iteration is accepted or rejected probabilistically where the probability depends on the change in value of the cost function resulting from such a perturbation. The terminology comes from observations in physical chemistry that certain chemical systems, particularly glass and metals, can be driven to low-energy states with useful physical properties by the process of annealing, which is a very gradual reduction in temperature. Simulated annealing as a combinatorial optimation technique comes from the Metropolis algorithm [237] for atomic simulations. The set of possible states is finite but the number of states is generally so large or astronomic that exhaustive search is not feasible.

Specifically, let $H(\mathbf{s})$ denote the cost function or "energy"' as a function of the state $\mathbf{s}$. In our case H is the average distortion for a nearest neighbor or Voronoi quantizer with a given codebook and the state $\mathbf{s}$ is the codebook or equivalently the set of labels associating each training vector with a particular cluster membership. In each iteration a candidate perturbation of the partition is randomly generated and then either accepted or rejected with a probability that depends on the cost of the new codebook. If the cost decreases, then the change is always accepted. If the cost increases it is accepted with probability

$$P = e^{-\beta \Delta H}$$

where $\beta = T^{-1}$, T is a parameter called the *temperature*, and ΔH is the increase in cost. If a sufficient number of iterations are made for a given temperature, the sequence of states approaches the condition of *thermal equilibrium* where the probability distribution of the states is stationary. If thermal equilibrium is reached at a very low temperature, then states that are globally optimal or very nearly so have very high probability. The time to approach thermal equilibrium increases with the temperature and so annealing is achieved by very gradually decreasing the temperature. Geman

and Geman [136] have shown that convergence in probability to a global minimum can be achieved with SA under the condition that the cooling schedule has temperatures decreasing in proportion to $(\log m)^{-1}$ where m is the iteration count. Generally, this cooling schedule is impractically slow and other faster schedules are chosen. Nevertheless, effective solutions have been found with this approach to many nonconvex optimization problems.

In VQ design from a training set, we indeed have a finite number of possible codebook designs since, as noted in the proof of Lemma 11.3.2, there is a finite number of ways to partition the training set into N clusters. The cost function is given by the distortion (11.3.6) incurred in coding the training set with the current codebook. Several reports of successful use of SA for VQ codebook design have been reported. In [319] SA is combined with the Lloyd iteration; the perturbation is implemented by randomly moving training vectors between neighboring partition regions until thermal equilibrium is reached, then a Lloyd iteration is performed before proceeding to the next temperature. In [119] the method of [319] is modified by omitting the Lloyd iteration and using a distortion measure that is more easily computed. In [52] the state perturbation is taken as a random perturbation to the codebook and and usual SA method of accepting or rejection the change is performed with a cooling schedule that is exponentially decreasing.

Fuzzy Clustering and Deterministic Annealing

It would be of great value if we could find a codebook design algorithm that leads to a global optimum without the typically very time consuming process that is introduced by the use of SA with effective cooling schedules. Recent work of Rose et al. [275] [276] [274] offers a novel approach to nonconvex optimization called *deterministic annealing* (DA), that appears to capture the benefits of SA for VQ codebook design without any randomness in the design process. In method, it is conceptually similar to the technique of *fuzzy clustering* described in [104] [33].

In designing a VQ codebook from empirical data, we encountered the formulas (11.3.4) and (11.3.5) for the centroid of a cluster (partition region) and and for the average distortion, respectively, where both were based on the selector functions. These functions can be regarded as *membership functions*, i.e., $S_j(\mathbf{v}_i)$ has value one if the training vector $\mathbf{v}_i$ is a member of cluster j and value zero otherwise. A cluster is said to be a *fuzzy* set if we may assign to each element of the training set a degree of membership or partial membership value between zero and one which indicates to what extent the particular vector is to be regarded as belonging to that set. In this way, we can generalize the selector functions to become partial

membership functions. Then, we define a *fuzzy distortion* to be

$$D_f = \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{N} d(\mathbf{v}_i, \mathbf{y}_j)[S_j(\mathbf{v}_i)]^q$$

where q is a parameter that controls the "fuzziness" of the distortion. If $q = 1$, then D_f is simply the average distortion where the contribution of each training set is weighted by its degree of membership in the cluster. In the latter case, the task of finding the best codebook and best partial memberships for the training vectors leads to the usual (nonfuzzy) solution. By choosing values of q greater than one a fuzzy solution is found. This suggests that a search procedure that begins with a large value of q and gradually reduces q to unity might lead to a global minimum. Actually no such procedure is known but DA is somewhat similar in concept.

In DA, instead of making random perturbations on the surface of a given cost function, the statistical description of the randomness (that would arise in a corresponding SA scheme) is incorporated into the cost function. A sequence of effective cost functions is parametrized by the temperature T. At infinite temperature the cost function will be convex; at high temperatures, the cost functions will be very smooth so that it is easy to move from one local minimum to another. As the temperature drops, the cost functions become more ragged and as $T \to \infty$, the effective cost function converges to the original cost function. In effect, DA finds the global minimum at a very high temperature and then tracks the changing location of this minimum as the temperature drops while using simpler convex optimization methods at each temperature. For brevity, we omit the mathematical description of DA.

11.4 Design Examples

We now turn to a variety of examples illustrating the generalized Lloyd algorithm. The examples range from artificially simple codes, which illustrate the basic operations, to codes for real sampled speech and image data. The quality of the basic "vanilla" VQ on such real sources is not high for very low bit rates. As we shall discuss in the remainder of the book, simple memoryless VQ is limited because of the rapid growth of complexity and memory with bit rate or dimension and large dimensions or rate are usually needed for high quality. Nonetheless the examples are important because they illustrate how the algorithm works and what quality is possible with basic VQ. All of the variations to be seen will still have the Lloyd algorithm at the core of their design. Hence they cannot be understood without first becoming familiar with the unadorned version.

IID Gaussian Source

The first example is a highly artificial one taken from [215] [153]. The source is a two-dimensional Gaussian vector with independent components each with zero mean and unit variance. As the training set for the design of a codebook containing $N = 4$ code vectors, twelve pairs of real numbers are selected at random from an iid Gaussian source having zero mean and unit variance. Thus, the training ratio is 3. As an initial guess for the codebook, we use a product code, taking a scalar quantizer with output levels $\{-1, +1\}$ for each of the two dimensions, that is, we use four points of a rectangular lattice in two dimensions. The training sequence is depicted by x's and the initial codewords by circles in Figure 11.4. The first step

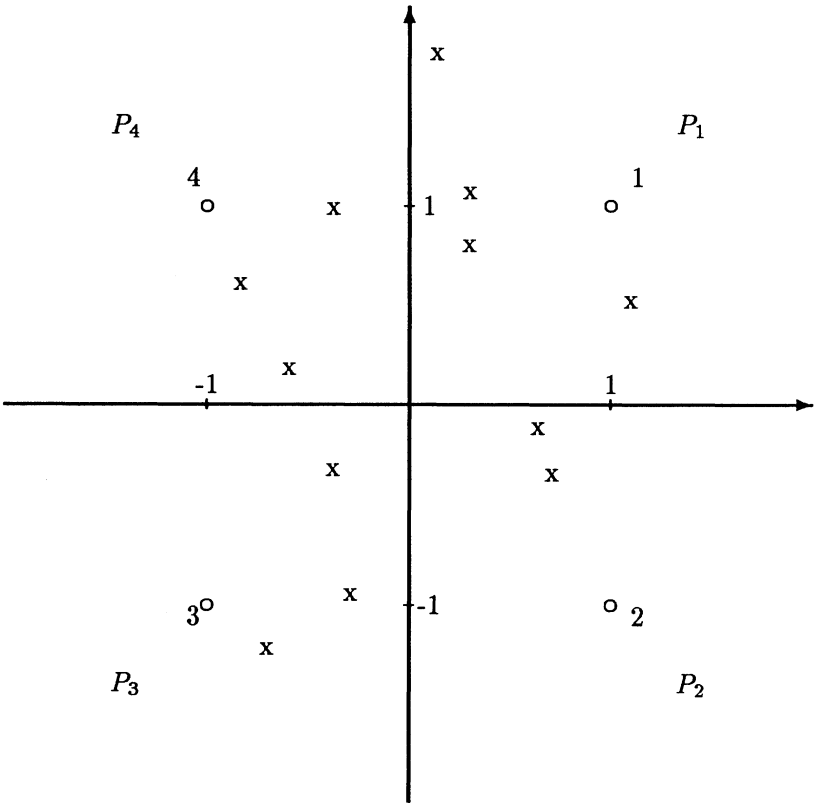


Figure 11.4: Training Sequence and Initial Code Book

of the Lloyd iteration is to partition the training sequence by performing a nearest neighbor encoding into the initial codebook. Assuming a squared

error distortion measure, this means assigning each training vector to the nearest code word, resulting in the partition corresponding to the four quadrants of the figure. Now that the training vectors are grouped, it is clear that the code words are not the best representatives for their groups. For example, in the upper right quadrant a good representative (in the sense of minimizing the squared error between the code word and the training vectors in the group) should lie within the cluster, near its visual center. The initial codeword is well outside of the cluster, producing a larger than necessary squared error. The second step of the Lloyd iteration fixes this: for each group of input vectors, the old code word is replaced by the group centroid, the vector which *by definition* minimizes the group's squared error. For the squared error distortion, this is the sample mean of the vectors in the group. The new code words are displayed in Figure 11.5. The code

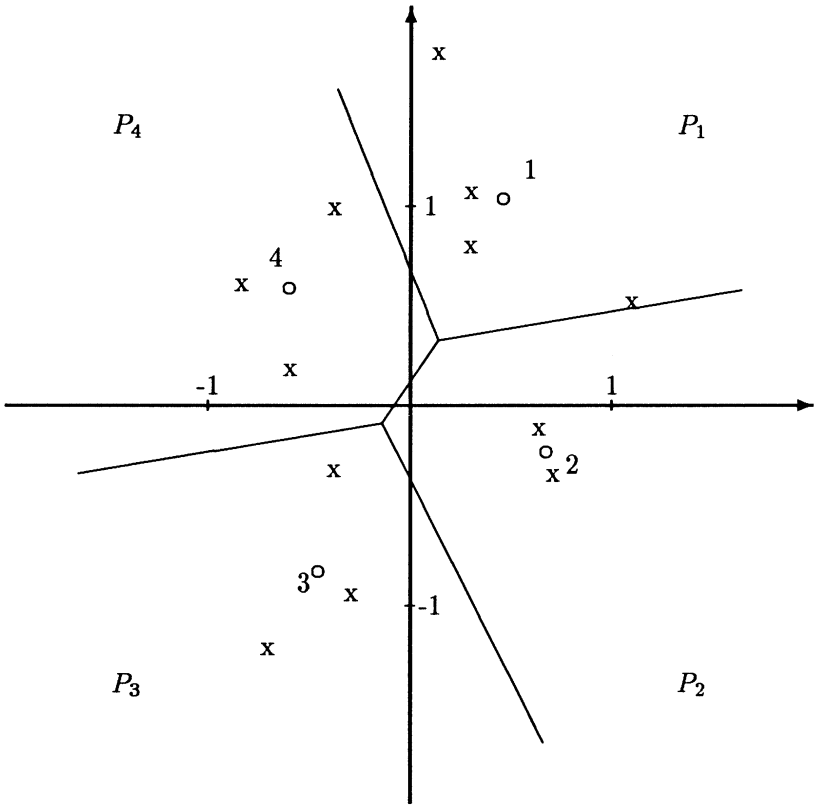


Figure 11.5: Centroids of Nearest Neighbor Regions

words have "moved" to the center of gravity of the vectors they are to

represent. This provides the first new codebook.

The quadrants now no longer correspond to the nearest neighbor regions of the new codebook. Hence the next Lloyd iteration begins by partitioning the space into nearest neighbor regions for the current codebook. These regions or cells are indicated by the thin lines.

In this simple example we are now done since the new cells are the nearest neighbor regions for the codewords, the codewords are the centroids of the cells, and there are no ties in the training set. Thus the Lloyd conditions are all satisfied.

Here we have only reported the average distortion of the code when it is used on the training sequence for which it is designed. If our intent were to design a good codebook and sell it, this would be cheating. Codes will often work unrealistically well on the training sequence. The real test is if they also work well on other data produced by the same source. We previously argued that if the training sequence is large enough, then the ergodic theorem implies that all of our sample averages are good estimates of the true expectations and that the training sequence numbers should be good indications of the performance on other long sequences. Here, however, our training ratio is ridiculously small and there is no such guarantee that the code will be robust. For the Gaussian examples we will focus only on the distortion incurred in quantizing the training set with the resulting codebook. This distortion is called for short the *design distortion* and is distinguished from distortions measured by testing performance on vectors that are not in the training set, the latter distortion is called *test distortion*. These two ways of assessing performance correspond to the commonly used phrases of coding "inside the training set" or "outside the training set," respectively. In the statistics literature, the performance inside the training set is called the "resubstitution" performance. Later we shall concern ourselves with test distortion when we turn to the more important real data sources such as speech and images.

A more complete design can be run on this simple case which illustrates the properties of a codebook produced by the Lloyd algorithm. Figure 11.6 shows the centroids and Voronoi (nearest neighbor) regions of a two-dimensional quantizer designed using the Lloyd algorithm on an iid zero mean unit variance Gaussian sequence. The codebook has 64 words so that the rate is $\frac{1}{2}\log_2 64 = 4$ bits per sample. The figure shows the polytopal shapes of the Voronoi regions with the codewords (centroids) (the dots) visually at the center of the regions.

As a more realistic example, the GL algorithm was used to design VQs of one bit per sample and dimensions $k = 1, 2, 3, 4, 5, 6$ (so that the codebook size is 2^k). The source is again assumed to have independent Gaussian components with zero mean and unit variance. Using a training sequence of

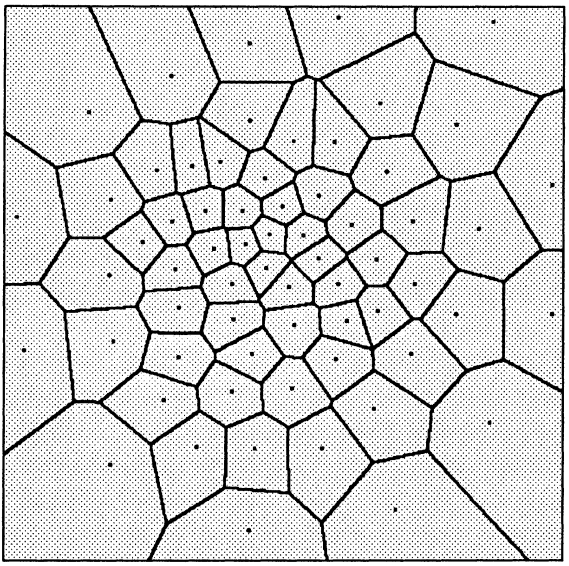


Figure 11.6: Voronoi Regions and Centroids: Two-Dimensional Gauss iid

100,000 samples and a distortion threshold of 0.01, the algorithm converged in fewer than 50 iterations and the resulting performance in terms of SNR vs. vector dimension is plotted as the solid line in Figure 11.7.

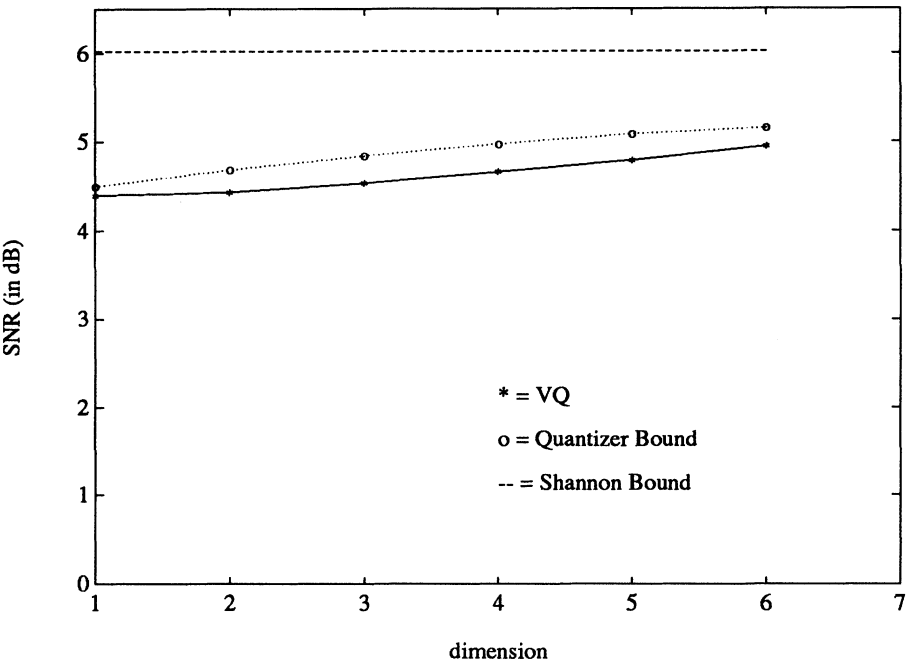


Figure 11.7: SNR vs. Blocklength: IID Gaussian

As a benchmark, two theoretical performance bounds are also plotted in the figure. The straight line corresponds to Shannon's distortion-rate function

$$D(R) = \sigma^2 2^{-2R},$$

where R is the rate in bits per sample [290] [134] [32] [159]. Here $R = 1$ and hence on a dB scale the distortion-rate function is $10 \log_{10} 0.25 = 6.02$ dB. The distortion-rate function provides an unbeatable lower bound to the average distortion of a rate R VQ of *any* dimension and arbitrary complexity. Furthermore, the Shannon bound is achievable in the limit as $k \to \infty$; that is, there exist codes that perform $D(R)$ (although Shannon theory does not say how to construct them). Clearly the performance of the actual VQs is not close to the optimum over all dimensions. A more realistic bound is

provided by the high resolution quantizer bound of (10.6.2), which is plotted in circles. This bound is a vector generalization of the high resolution approximation for scalar quantizers of Chapter 5. Although strictly speaking these bounds are only accurate for large resolution, they are reasonably close to the actual performance and can be trusted at larger dimensions, e.g., when $k = 6$ and hence $N = 64$.

In this example a natural way of generating initial guesses for dimension k is to use the product code (to be defined more carefully in the next chapter) formed by using the $k-1$ dimensional quantizer just designed for the first $k-1$ samples and a one bit scalar quantizer on the final coordinate. This provides a one bit per sample dimension k codebook which provides a good start for the dimension k codebook. If, however, one only wishes to design a dimension k codebook and has no need for smaller dimensions, then the splitting algorithm works well.

Gauss Markov Sources

We have previously argued that one might expect VQ to show more improvement in comparison with scalar quantization when the source has significant memory. A simple example of a source with memory and a common model for a variety of sources in signal processing applications is the Gauss Markov or Gauss autoregressive source $\{X_n\}$ defined by

$$X_{n+1} = aX_n + W_n,$$

where the regression coefficient a has magnitude less than 1 and where $\{W_n\}$ is a zero mean, unit variance, iid Gaussian source. (The variance is chosen as unity for convenience; the extension to arbitrary finite variance is straightforward.) The autocorrelation of this source has the familiar form

$$R_X(k) = \frac{a^{|k|}}{1-a^2}; \quad \text{all integers } k.$$

We assume $a = 0.9$, in which case Shannon's bound to optimal performance (over all block lengths) is 13.2 dB.

Vector quantizers were designed for this source at a rate of 1 bit per sample based on a training sequence of 60,000 samples and dimensions of 1 to 7. The performance is plotted in Figure 11.8. The improvement over scalar quantization is now more striking as expected. The codes were also tested on other sequences outside of the training sequence (but produced by the same random number generator) and the differences were under 0.25 dB, suggesting that the training sequence is large enough in this example to be trustworthy. For comparison, Arnstein's optimized one bit per sample

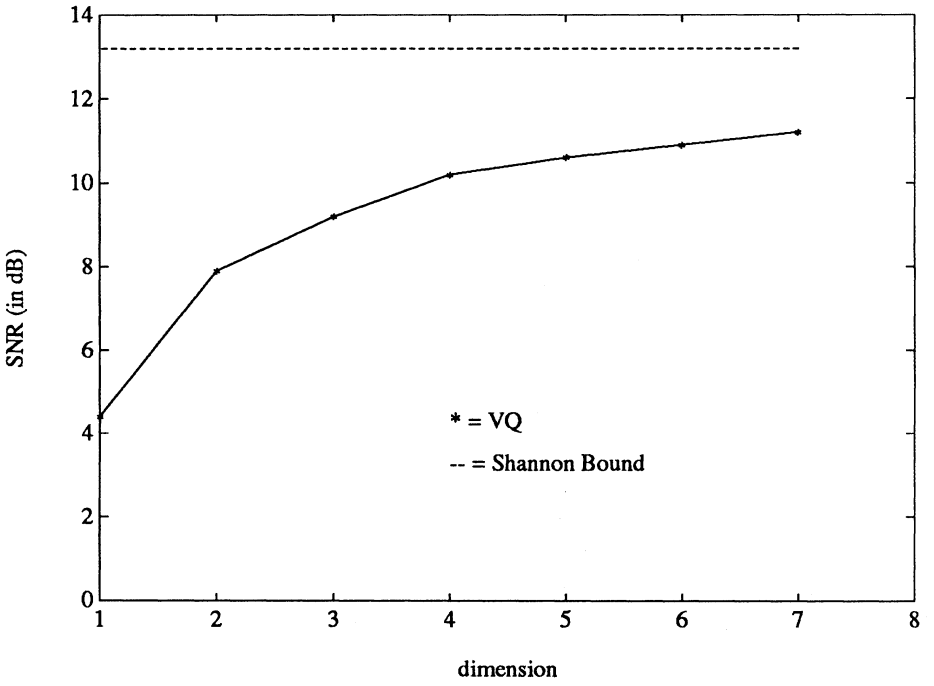


Figure 11.8: SNR vs. Blocklength: Gauss Markov Source

predictive quantization system [18] yields an SNR of 10 dB, close to that of a dimension 4 VQ. Three additional dimensions provide approximately 1 dB further improvement. We shall later see that a vector predictive quantizer can outperform the dimension 7 VQ by using only dimension 2 vectors.

Clearly the same basic approach works for any Monte Carlo generated random process and several papers have appeared reporting the performance of VQs on a variety of sources, especially on Laplacian and gamma densities. A variety of results along these lines may be found in the IEEE reprint collection [2].

Speech Waveform Coding

As a first source of practical importance we consider sampled speech. We should begin with the admission that plain VQ does not provide outstanding quality when applied directly to speech; more sophisticated VQs are required. It does, however, provide some interesting comparisons with traditional techniques and a straw dog for comparisons with smarter systems later on. Another caveat is that squared error is a notoriously poor measure of subjective quality for speech, but is often used anyway because of its simplicity. Good quality systems commonly use weighted squared error distortion measures, where the weightings are in the frequency domain and reflect knowledge of the human auditory system.

A training sequence of 640,000 samples of ordinary speech from four different speakers sampled at 6.5 kHz was generated. One and two bit per sample codes were designed in exactly the same way as those for the Gaussian sources. Figure 11.9 presents the SNRs for a test sequence consisting of 76,800 samples from a speaker not in the training sequence. The improvement in performance with dimension is largest at the smaller dimensions, but maintains a slope of 0.6 dB (per dimension) for dimensions 5 and above. The two bit per sample codes were designed for dimensions 1 through 4. These codes were designed using the splitting technique.

It is worth observing at this point that the SNRs reported in such experiments are of primary interest for relative comparisons among different code design techniques run on the same training set and compared on common test sets. For example, the SNRs reported here are uniformly lower by from 0.5 dB to 2 dB than those reported in [139] for basic vector quantization of speech waveforms. In addition to using different training and test sets, the sampling rate of the speech waveform results reported here was 6.5 kHz in comparison with the 8 kHz of [139]. The higher the sampling rate, the less variable are the vector shapes for the same dimension and the simpler the needed codebooks. (This is essentially the idea behind oversampling analog-to-digital conversion.)

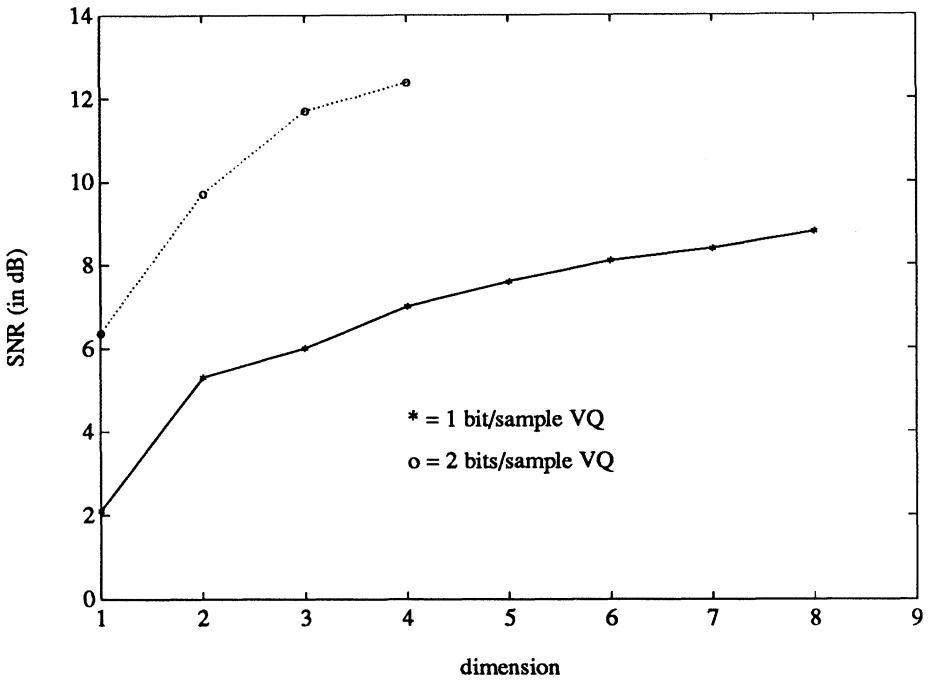


Figure 11.9: SNR vs. Blocklength: Sampled Speech

To gain a feeling for the SNR comparison with traditional techniques, the two bit per sample codes can be compared with the Lloyd-Max scalar quantizer design based on a gamma model density yielding an SNR of 6.04 dB on real speech [249]. The Lloyd algorithm gave slightly better performance of 6.4–6.5 dB for a scalar quantizer, but significantly better performance (more than 5 dB better) at higher dimensions. A scalar Lloyd-Max quantizer optimized for a Laplace model and imbedded in a predictive quantization system led to an SNR of 11.67 dB, a performance slightly lower than with a 3 dimensional VQ. The low rate Karhunen-Loeve transform coder of Campanella and Robinson [44] results in an SNR of 12.6 dB at 14 kbits/sec (their Fourier transform coder is about 4 dB worse). They used a transform vector dimension of $k = 16$. Their performance is close to that of the dimension 4 VQ. The adaptive predictive PCM (ADPCM) system of Crochiere et al. [83] yielded a performance of 10.9 dB at 2 bits per sample and of 11.1 dB for their subband coder of the same rate, comparable to the dimension 3 VQ. (Of course the subband coder has some perceptual advantage not reflected by SNR values.) The point of these comparisons is that simple memoryless VQ is at least competitive with several traditional scalar-quantizer based systems employing prediction or transforms in order to increase the effective vector size.

Simulated Speech

Unlike the Gaussian examples previously considered, the performance of VQ on speech cannot be compared to theoretical optima because no such optima are known. A natural possibility is to compute the optimal performance for random process models of speech, that is, well-defined random processes that exhibit similarity with speech. This is of interest because it might both suggest performance bounds and indirectly shed light on how good such models of real data sources are. It is also of interest in the Gaussian case because it shows even better than the Gauss Markov source how the performance improvement with dimension increase can be greater for more highly correlated sources. For this experiment an autoregressive or all-pole model for speech was used (see Section 2.8 for the terminology and Section 4.10 on random process modeling). This model is a gross simplification of reality since speech is known to exhibit a far more complex statistical behavior. Nevertheless, autoregressive modeling is very commonly used for modeling the short term behavior of speech for an individual phonetic unit whose duration is a small fraction of a second. More sophisticated models use a different autoregressive model for successive segments.

Here we model speech as a stationary random process $\{X_n\}$ of the form

$$X_n = \sum_{k=1}^{M} a_k X_{n-k} + \sigma W_n,$$

where the W_n are produced by an iid zero mean random process and the parameters a_k are chosen so that the filter with z-transform

$$\frac{1}{A(z)} = \frac{1}{1 + \sum_{k=1}^{M} a_k z^{-k}}$$

is stable. The parameter σ^2 is the squared gain or one-step prediction error of the process $\{X_n\}$ as developed in Chapter 4. An autoregressive source having the same autocorrelation as a speech sequence to lag M is called a *matching autoregressive source* or *MARS* for speech [12]. The same sampled speech training sequence with $L = 640,000$ can be used to estimate the autocorrelation function of speech to a lag of M by computing the sample autocorrelations

$$r(k) = \sum_{l=1}^{L-k} x_l x_{l+k}$$

for $k = 1, 2, \cdots, M$. The results are described in Table 11.2 where the values $r(k)$ are normalized by the energy $r(0) = 9662.57$. It can be shown that

Lag k	$r(k)/r(0)$	a_k
0	1.00000	1.00000
1	0.40094	-0.35573
2	-.04745	0.29671
3	-0.16734	0.09716
4	-0.18298	0.13798
5	-0.19850	0.10661
6	-0.26476	0.18392
7	-0.25939	0.17867
8	-0.12955	0.07244
9	-0.01075	0.11004
10	0.10457	0.03551

Table 11.2: Speech Autocorrelation and Filter Parameters

because of the autocorrelation matching property of linear prediction [234],

the filter parameters a_k which yield an Mth order autoregressive process having exactly the given autocorrelation to lag M are the a_k produced by the Levinson-Durbin algorithm used in Chapter 4 to find the optimal linear predictor based on the finite past for a process given M autocorrelation values. Intuitively, if one can build a good predictor for a process, then one also can construct a good model for the process. Also intuitively, if $P(z) = -\sum_{k=1}^{M} a_k z^{-k}$ is the z-transform of a good finite impulse response linear predictor for a process, then ideally the error that equals the difference between the original process and its linear prediction, that is, the signal produced by passing the original process through a linear filter with z transform $A(z) = 1 + \sum_{k=1}^{M} a_k z^{-k}$, should be white. (Recall that the prediction coefficients were the negative a_k.) Thus if one wants to produce a process with the same autocorrelation function as the original process, one can reverse the system and drive the inverse filter (an autoregressive filter) $1/A(z)$ with a white process. See the discussion of Section 4.10 on simulation of random processes for further details. This produces an output with power spectrum $1/|A(e^{-2\pi jf})|^2$, which turns out to have an autocorrelation function which agrees with the given autocorrelation function. Thus we can run the Levinson-Durbin algorithm on the autocorrelation of Table 11.2 to find the filter parameters of Table 11.2. This filter will arise later when considering the LPC-VQ vocoder. The remaining issue is how to choose the probability density function for the iid process which drives the model. We choose a popular density function for this purpose, the Laplacian or doubly exponential density given by

$$f_Z(z) = \frac{1}{\sqrt{2}} e^{-\sqrt{2}|z|}.$$

Figures 11.10 and 11.11 depict the performance of VQs designed for real speech and for a training set of 60,000 samples from the matching autoregressive source with the Laplacian driver (produced by a random number generator). The training set for the fake speech is relatively short and the reported distortions are design distortions rather than independent test distortions. This use of design distortion was simply for convenience because these early experiments were performed several years ago on relatively small and slow computers. Nevertheless, they still provide an interesting comparison provided one realizes that the reported design distortions are generally optimistic. Also plotted are the high resolution quantization bound of (10.6.2) and Shannon's distortion rate bound. (Details of the bounds as well as similar experiments for K_0 marginals may be found in [3]. The distortion rate bounds are treated in [159].)

Several observations are worth making. First, the relative performance improvements achieved by real speech are much larger than those achieved

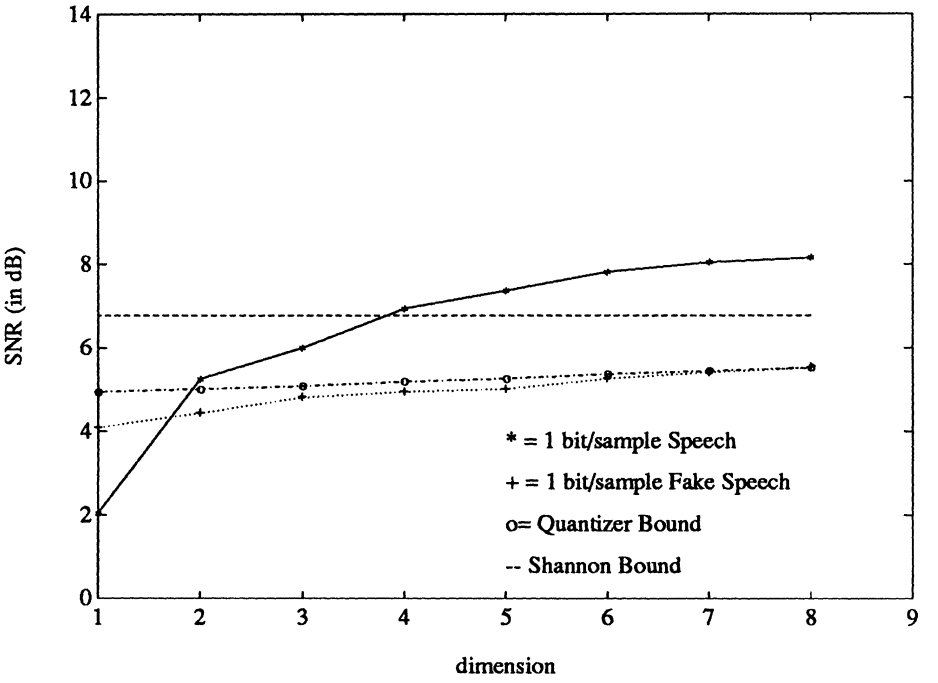


Figure 11.10: SNR vs. Blocklength: Speech and Fake Speech at 1 b/s

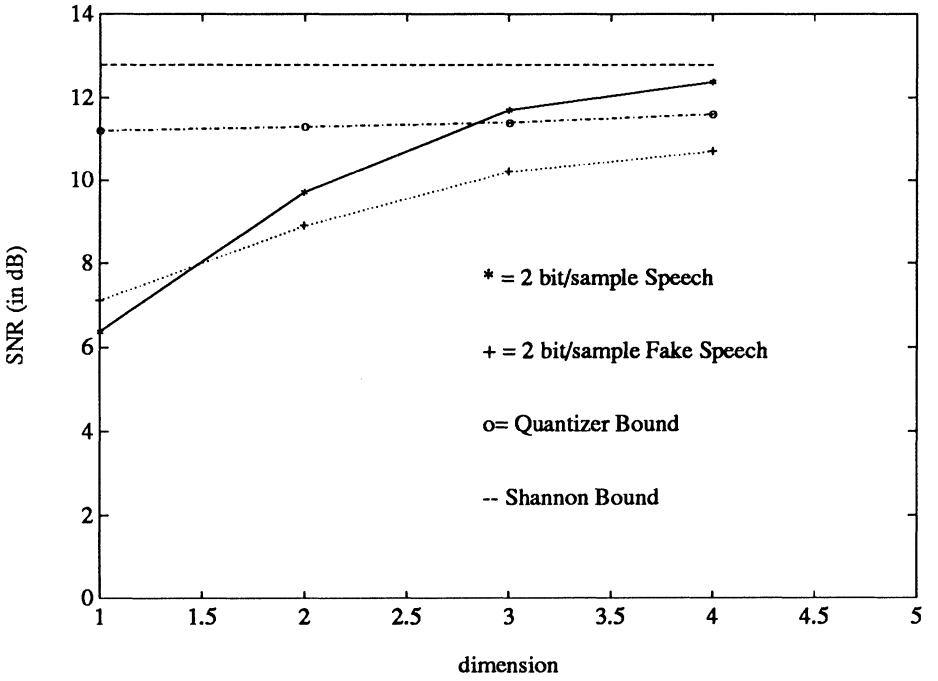


Figure 11.11: SNR vs. Blocklength: Speech and Fake Speech at 1b/s

for the model. This suggests that real speech has a much richer structure than the model and that a code trained directly on the real speech is more likely to capture and use this structure than is a code designed on a model of speech. Second, the actual performance of the designed VQs is better than the theoretical Shannon "optimum" in the 1 bit per sample case. This is not a violation of theory, the bounds are based on the model, not on real speech. It again suggests that the model is not very good, especially not at predicting the performance for real speech. Third, the performance on the model compares quite well to the high resolution quantization bound at the higher dimensions (and hence the higher rates in bits per vector). This suggests that in some applications at realistic dimensions VQ can perform nearly optimally for a fixed dimension and rate. Note in particular that the highly correlated speech model has performance much closer to the theoretical bound than the less correlated Gauss Markov and iid Gauss sources.

We should not leave speech waveform coding without commenting on the subjective quality of simple VQ. For a rate of 1 bit per second (b/s) the quality goes from very noisy and only barely intelligible scalar codes (effectively hard limiters) to easily intelligible but still noisy at dimension 8. The quality improves steadily with increased dimension. For 2 b/s, the quality is significantly better at all dimensions. A dimension 4, 2 b/s code exhibits little quantization noise on an ordinary high fidelity audio tape system. When these tests are run on a test sequence outside of the training sequence, there is a clear increase in quantization noise and buzziness, especially at the smaller dimensions and 1 b/s.

LPC-VQ

An alternative to coding a speech waveform directly is to construct a model of a short segment of speech, say 25 ms or so, quantize the parameters of the model, and then synthesize the reproduction using the quantized model parameters. Although this analysis/synthesis approach to speech compression is quite old, its major impact for good quality practical systems came with the development of the PARCOR or linear predictive coding (LPC) approaches to analysis/synthesis due to Itakura and Saito [189] [190] and Atal and Hanauer [19]. The basic idea is to model speech as an all-pole or autoregressive filter driven by either a white noise source (for unvoiced sounds such as "sss" and "sh") or a periodic pulse train (for voiced sounds such as vowels). The parameters describing the filter, voicing pitch and period are quantized and communicated. The user then synthesizes the speech by simulating the filter and driver. The technique, commonly called LPC, is treated in detail in numerous texts such as [234] [195] [247]. Here

we extract a few key points necessary to describe the combination of LPC and VQ. The name "linear predictive coding" is appropriate because, as noted previously, the construction of an all-pole model for speech turns out to be equivalent to the construction of an optimal one-step finite memory linear predictor for speech. The model can be designed by first estimating the sample autocorrelation of the speech and then running the Levinson-Durbin algorithm to obtain the filter parameters describing the predictor or, equivalently, the model. The voiced/unvoiced decision and the determination of pitch are completely separate (and difficult) issues and will be ignored here but are amply discussed in the literature. See for example [247].

Traditional LPC systems used scalar quantization of the various parameters, although a variety of distortion measures, parameter representations, and bit allocation strategies were used. (See, e.g., [171].) If a VQ is to be used, clearly a distortion measure is required and it is not clear that the squared error is appropriate. An alternative and natural distortion measure was implicit in Itakura and Saito's original development. Rather than using linear prediction arguments, they used techniques from estimation theory to show that the selection of the best autoregressive model given a vector (or frame) of speech could be viewed as a minimum distortion selection procedure using what they called the "error matching measure." The error matching measure has since come to be known as the Itakura-Saito distortion. Suppose that we consider the input vector $\mathbf{x}$ to be a vector of N speech samples, typically from 50 to 200 samples. The reproduction to be produced by the vector quantizer however, will not be a quantized speech sequence. Instead it will be an Mth order ($M << N$) autoregressive model for speech of the form $\hat{\mathbf{x}} = \sigma/A(z)$ with

$$A(z) = 1 + \sum_{k=1}^{M} a_k z^{-k}.$$

Typically N is from 8 to 15. The final reproduced speech segment will then be synthesized by using this filter on either zero mean white noise or a periodic pulse train. In fact, the filter is determined by linear prediction arguments (Atal and Hanauer) or maximum likelihood arguments (Itakura and Saito), both of which lead to running the Levinson-Durbin algorithm on the speech frame sample autocorrelation (for the autocorrelation method of LPC). In principle, however, this is equivalent to choosing $\sigma/A(z)$ (or any equivalent set of parameters) over all stable, minimum phase filters so as to minimize the distortion measure

$$d(\mathbf{x}, \hat{\mathbf{x}}) = \frac{\mathbf{a}^t \mathbf{R}(\mathbf{x}) \mathbf{a}}{\sigma} - \ln \frac{\alpha_M(\mathbf{x})}{\sigma} - 1, \tag{11.4.1}$$

where $\mathbf{a}^t = (1, a_1, \cdots, a_M)$, where

$$\mathbf{R}(\mathbf{x}) = \{r_x(k-j);\ k, j = 0, 1, \cdots, M\}$$

is the $(M+1) \times (M+1)$ matrix of sample autocorrelations from 0 to M lag for the input sequence of N samples, that is,

$$r_x(k) = \frac{1}{N-k} \sum_{j=0}^{N-k-1} x(j)x(j+k).$$

Finally,

$$\alpha_M(\mathbf{x}) = \min_{\mathbf{y}:b_0=1} \mathbf{b}^t\mathbf{R}(\mathbf{x})\mathbf{b} \tag{11.4.2}$$

is the quantity D_M of the Levinson-Durbin algorithm of Section 4.5, the minimum mean squared prediction error achievable by a linear predictor with M stages. That d is nonnegative and hence is indeed a distortion measure follows from the $\ln x \leq x - 1$ inequality and (11.4.2) since

$$\begin{aligned} \frac{\mathbf{a}^t\mathbf{R}(\mathbf{x})\mathbf{a}}{\sigma} - 1 - \ln\frac{\alpha_M(\mathbf{x})}{\sigma} &\geq \ln\frac{\mathbf{a}^t\mathbf{R}(\mathbf{x})\mathbf{a}}{\sigma} - \ln\frac{\alpha_M(\mathbf{x})}{\sigma} \\ &= \ln\frac{\mathbf{a}^t\mathbf{R}(\mathbf{x})\mathbf{a}}{\alpha_M(\mathbf{x})} \\ &\geq 0 \end{aligned}$$

The distortion of (11.4.1) appears strange at first: it is not symmetric and is not simply related to a metric. The asymmetry should not cause a problem since the input and output are very different; one is a sequence of real sampled speech and the other describes a model for synthesizing speech. The distortion measure is called the *modified Itakura-Saito distortion* or often simply the *Itakura-Saito distortion* without the extra adjective. Strictly speaking, the original Itakura-Saito distortion, which they called the "error-matching measure" [189][190], has α_∞, the infinite one-step prediction error σ_p^2 of Section 4.9 in place of $\alpha_M(x)$ in (11.4.1). The modification yields an equivalent nearest neighbor rule and is computationally simpler (See, e.g., [165].)

The Itakura-Saito distortion has many interesting properties for computation, theory, and applications [161] [296] [165]. For example, it can be expressed in terms of the sample power spectrum of the input sequence and the power spectrum of the autoregressive model. It can be shown to be a special case of a relative entropy or Kullback-Leibler information rate. It can be put into a more computationally transparent form by writing

$$\mathbf{R}(\mathbf{x}) = \{r_x(k-j);\ k, j = 0, 1, \cdots, M\}$$

and by defining the inverse filter parameter autocorrelations

$$r_a(k) = \sum_{l=0}^{M} a_l a_{l+k}$$

and then doing some matrix algebra to write

$$d(\mathbf{x}, \hat{\mathbf{x}}) = \frac{\alpha}{\sigma} - \ln \frac{\alpha_M(\mathbf{x})}{\sigma} - 1, \tag{11.4.3}$$

where

$$\alpha = r_x(0) r_a(0) + 2 \sum_{k=1}^{M} r_x(k) r_a(k), \tag{11.4.4}$$

an inner product between the autocorrelation of the input sequence (up to M lags) and the autocorrelation of the inverse filter coefficients.

Since the Itakura-Saito distortion measure is implicit in the selection of the perfect (unquantized) all-pole model for speech, it is reasonable to use the same distortion measure when selecting one of a finite number of possible reproductions, that is, when using a VQ which has a speech sequence as an input and an all-pole model as an output. In order to use the Lloyd algorithm, however, one must be able to find a centroid with respect to a distortion measure. Recall that for any collection $R = \{\mathbf{x}_1, \cdots, \mathbf{x}_K\}$, where $K = ||R||$, we must be able to find a vector $\mathbf{y} = \sigma/A(z)$ which minimizes the sample distortion

$$\sum_{i=1}^{K} d(\mathbf{x}_i, \mathbf{y}) = \sum_{i=1}^{K} [\frac{\mathbf{a}^t \mathbf{R}(\mathbf{x}_i) \mathbf{a}}{\sigma} - \ln \frac{\alpha_M(\mathbf{x}_i)}{\sigma} - 1].$$

Equivalently, we must find the $\mathbf{y}$ that minimizes the above sum normalized by $1/K$, and hence we must find the σ and the $\mathbf{a}$ which minimize

$$\frac{\mathbf{a}^t \bar{\mathbf{R}} \mathbf{a}}{\sigma} - \frac{1}{K} \sum_{i=1}^{K} \ln \frac{\alpha_M(\mathbf{x}_i)}{\sigma} - 1 \tag{11.4.5}$$

where $\bar{\mathbf{R}}$ is the average sample autocorrelation

$$\bar{\mathbf{R}} = \frac{1}{K} \sum_{i=1}^{K} R(\mathbf{x}_i).$$

Only the leftmost term of (11.4.5) depends on $\mathbf{a}$ and its minimization over all $\mathbf{a}$ with $a_0 = 1$ is exactly the minimization encountered in Chapter 4 (in

(4.3.12)). It is solved by the Levinson-Durbin algorithm which gives the $\mathbf{a}$ vector portion of the centroid. If $\overline{\sigma}$ is the resulting minimum of $\mathbf{a}^t\overline{\mathbf{R}}\mathbf{a}$, then the σ minimizing (11.4.5) is simply $\sigma = \overline{\sigma}$ since, ignoring terms not depending on σ, we have that

$$\frac{\overline{\sigma}}{\sigma} + \ln \sigma \geq 1$$

with equality if $\sigma = \overline{\sigma}$. Note that in order to find the nearest neighbor and the centroids with respect to the Itakura-Saito distortion, we do not actually have to compute the individual $\alpha(\mathbf{x}_i)$ for the input vectors, a computation which can be time consuming. It is necessary, however, to compute these mean squared errors if one wishes to quantitatively measure the Itakura-Saito distortion.

We now have all the pieces: to design an LPC-VQ we run the Lloyd algorithm (here the splitting method is a natural initialization) on the speech training sequence. The same basic training sequence is first parsed into long vectors (frames) of 128 samples each. Each of these vectors has its sampled autocorrelation matrix $\mathbf{R}(\mathbf{x}_i)$ computed and this can be considered henceforth as the training sequence. Here the splitting algorithm is natural. One first finds the centroid of the entire training sequence. In fact we have already done this in the previous example when we constructed the simulated speech model by finding the autoregressive source that matched the speech autocorrelation to lag 10. This model is split into two close models (e.g., just change the gain term σ slightly) and the training sequence is encoded into one of the two reproduction vectors using the Itakura-Saito distortion. Note that we do not need to compute the entire distortion to find the best fit. For each input vector $\mathbf{x}$ we need only find the model for which the σ and $\mathbf{a}$ minimize

$$\frac{a^t\mathbf{R}(\mathbf{x})a^t}{\sigma} + \ln \sigma;$$

that is, we need not evaluate all the $\alpha_M(\mathbf{x})$ terms. After partitioning the input space, we replace the codewords by the centroids, found using the Levinson-Durbin algorithm. We then continue to convergence at a rate of 1 bit per vector, split, and so on. The results are depicted in Figure 11.12. Shown are the rate in bits per vector (128 input symbols) and the design and test SNRs (here ten times the logarithm of the ratio of the Itakura-Saito distortion of the code to that of the optimal zero rate code.) Since the number of samples is fixed and the vector size has increased, the training sequence now has only 5000 vectors in it. The test sequence has 600 vectors. Note that the design performance is noticeably optimistic, often excelling the test performance by 1 dB or more.

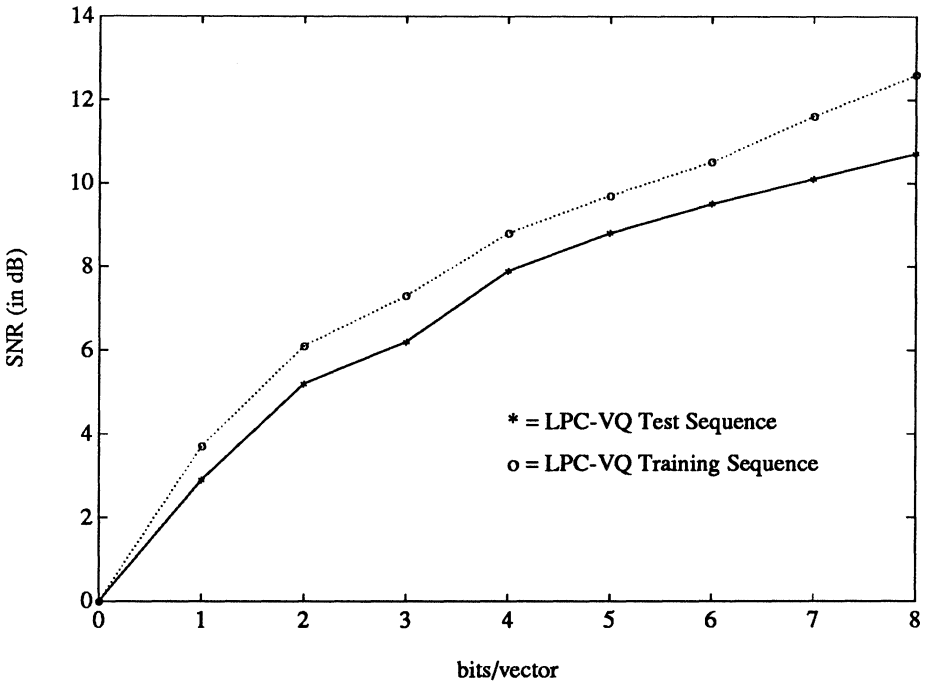


Figure 11.12: SNR vs. Bits per vector: LPC-VQ

The LPC VQ was one of the earliest successful applications of VQ because it brought the bit rate of traditional LPCs using scalar quantization for various parameterizations of the LPC coefficients from 4800 bits per second down to 1200 bits per second with little loss of quality. The problem remained that LPC itself has questionable quality, often tinny and sensitive to high and low pitched voices as well as multiple speakers. Nonetheless, LPC VQ forms a core part of most of the best current low to very low rate (2.4 kb/s or below) voice coding systems. Several of these variations will be sketched later. The preceding example, based on the Itakura-Saito measure, is just one of many alternative methods available today for coding the linear prediction parameters. Efficient coding of linear prediction coefficients has become an important issue in almost all advanced speech coders studied today since the autoregressive model is widely used in waveform coding of speech as well as in low rate vocoders.

Image VQ

A natural way to apply basic VQ to images is to decompose a sampled image into rectangular blocks of fixed size and then use these blocks as the vectors. For example, each vector might consist of a 4×4 square block of picture elements or *pixels*. Thus each vector has 16 coordinates. A typical digital image has a resolution of 8 bits per pixel (bpp). The goal of VQ is to reduce this to less than 1 bit per pixel without perceptible loss of picture quality. We shall see that this is not possible with basic VQ, but will be possible with some variations. The application of VQ to coding images was independently proposed in [344], [140], and [23] and subsequently research activity in this area expanded greatly. The earliest study, by Yamada, Fujita, and Tazaki [344], appeared in Japanese in 1980. Subsequently commercial video teleconferencing coders emerged in the U.S. and Japan based on VQ.

We shall consider two different types of images for variety. The first consists of ordinary still frame images, for which we will use a training and test set taken from an image data base prepared at the University of Southern California. Actually the USC image data base comes in full 24 bpp color, but we here consider only gray scale compression by taking only the luminance component of the image, that is, the so-called Y component of the YIQ transformation from the red, green, blue components of the original color image. Several of the images from this data base are shown in Figure 11.13. The lower right image, called variously "Lenna" and "the girl with the hat," was not in the training sequence and is used as our test image for this data base.

The Lloyd algorithm was run on the training sequence using the simple

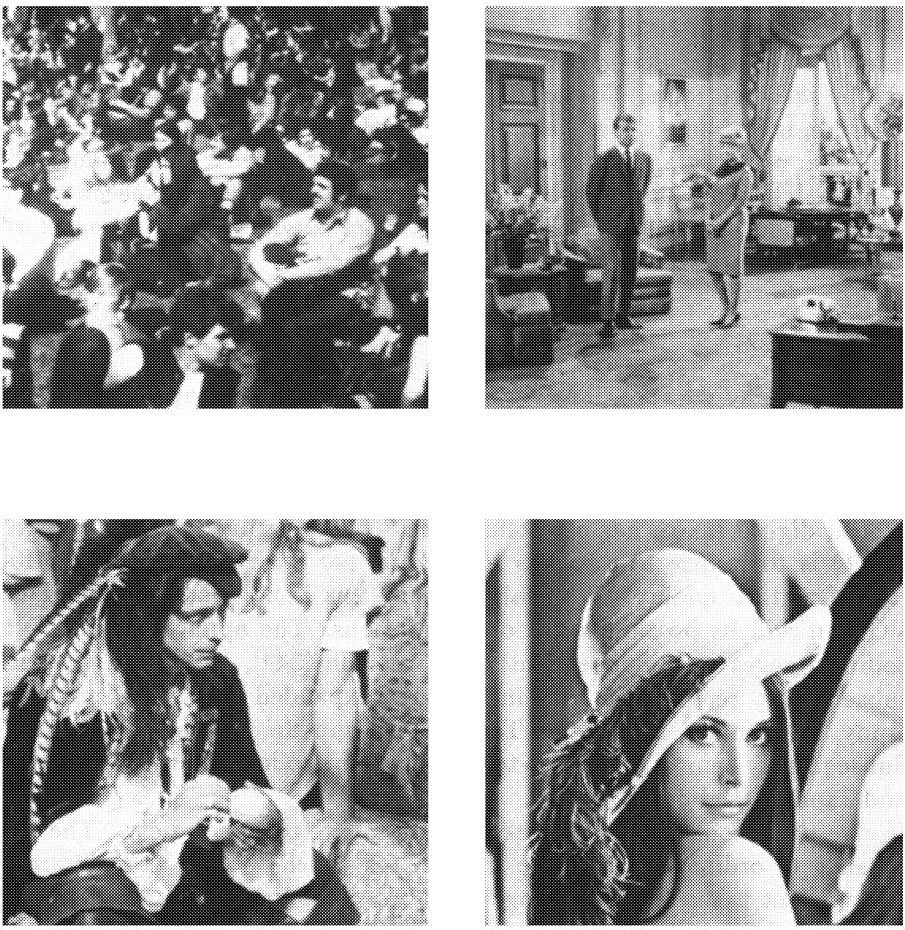


Figure 11.13: USC Images

squared error distortion measure and 4×4 blocks. The quality measure for ordinary still images is commonly taken as the peak signal-to-noise ratio or PSNR which is defined as 10 times the log to the base 10 of the ratio of peak input amplitude squared to the mean square error (MSE). In the case of an 8 bpp original this is

$$\text{PSNR} = 10 \log_{10}(\frac{255^2}{\text{MSE}}).$$

The PSNR is plotted as a function of the rate in bits per pixel (bpp) in Figure 11.14.

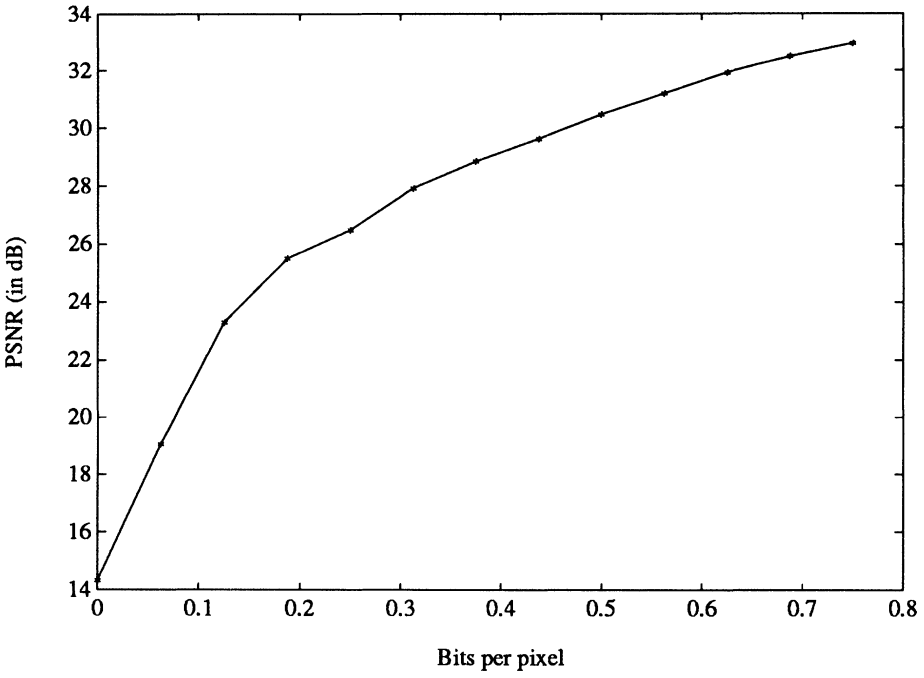


Figure 11.14: PSNR vs. Bits per pixel: Lenna

The resulting codebook for .5 bits per pixel (bpp) is shown in Figure 11.15. There are 256 different patterns (8 bits) of 4×4 pixel squares so that the rate is indeed $8/15 = 0.5$ bpp.

In order to get a feeling for the corresponding subjective quality and the operation of the code, Figure 11.16 shows the coded Lenna at 0.5 bpp and Figure 11.17 shows the resulting error image, the difference between the

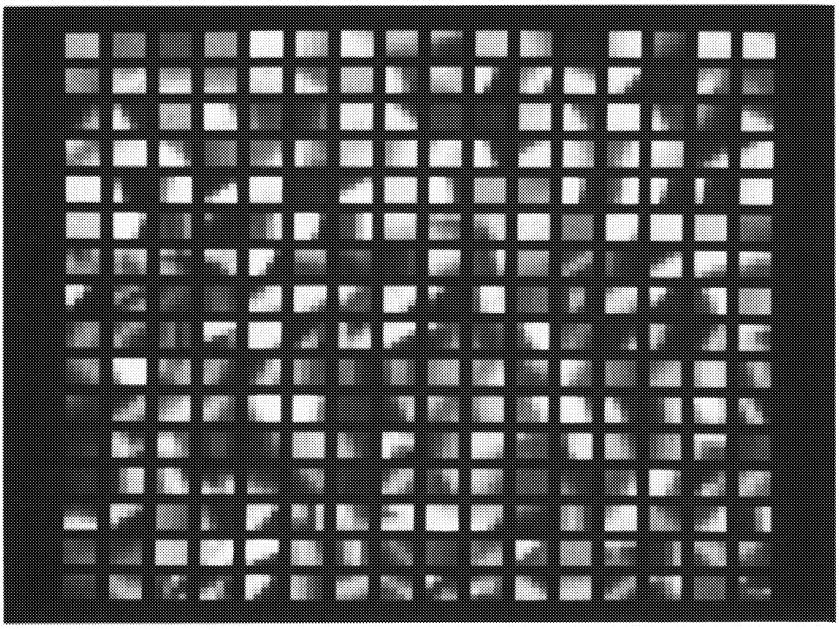


Figure 11.15: Image VQ Code Book

Figure 11.16: Lenna at 0.5 bpp

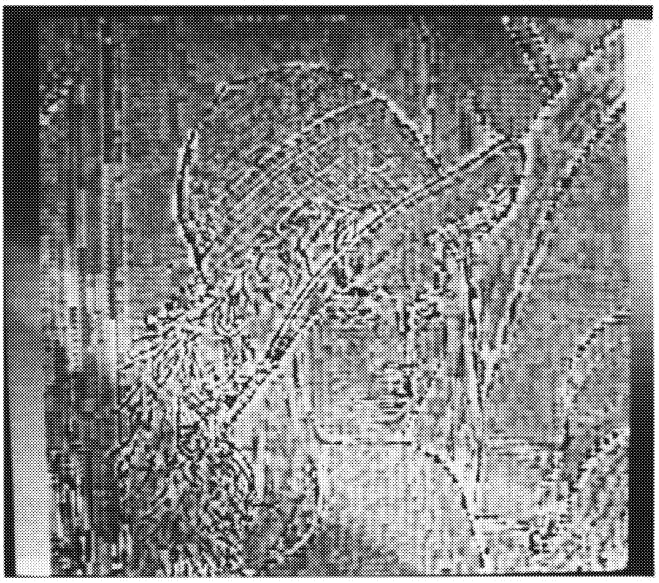


Figure 11.17: Quantization Error

original and the coded image. Figure 11.18 shows a closeup of the eye of the original Lenna image along with the coded eye. One can see that even the original is not perfect when magnified enough; high resolution digital images are blocky when viewed at this scale. The VQ image is, however, much worse: the blockiness and the sawtooth effect along diagonal edges are apparent. Figure 11.19 shows a similar closeup of the original image along with the coded version of the shoulder. This shows the inability of a simple VQ to well code diagonal edges.

As a distinct image example, a training sequence of magnetic resonance (MR) images of sagittal brain slices were used to design a 1 bit per pixel (bpp) VQ with 4×4 blocks. The original 256×256 image was 8 bpp and the Lloyd algorithm was used to design a VQ. (In fact the original MRI images produced had 9 bits, but only the highest order bit was truncated for this study because our monitor could only display to 8 bit accuracy.) The original and VQ image of a person not in the training sequence (but the same view) are shown in Figure 11.20. We refer to this image as "Eve" as it shows the brain of Dr. Eve Riskin who did much of the original work applying vector quantization to medical images. Once again the block and staircase effects can be seen in the edges, but the overall appearance is fairly good for a simple code. When dealing with medical images, ordinary

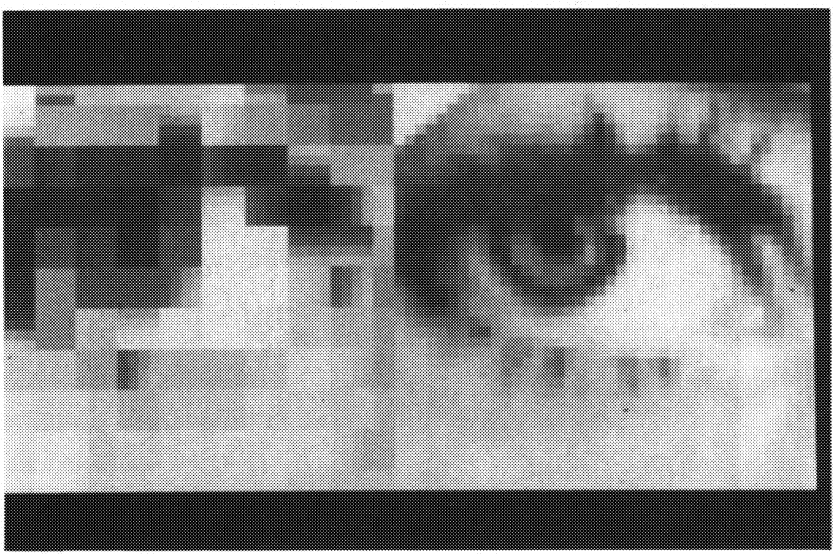


Figure 11.18: VQ Coded Eye and Original

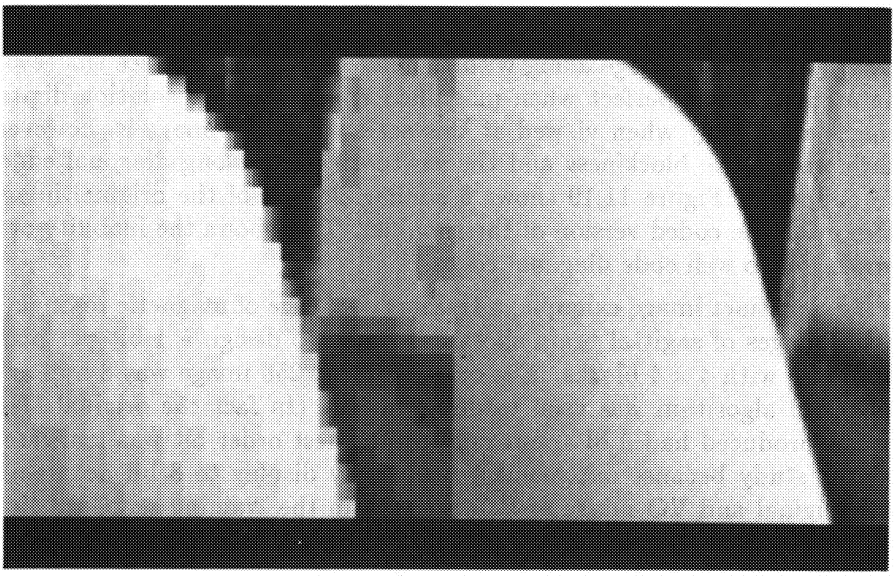


Figure 11.19: VQ Coded Shoulder and Original

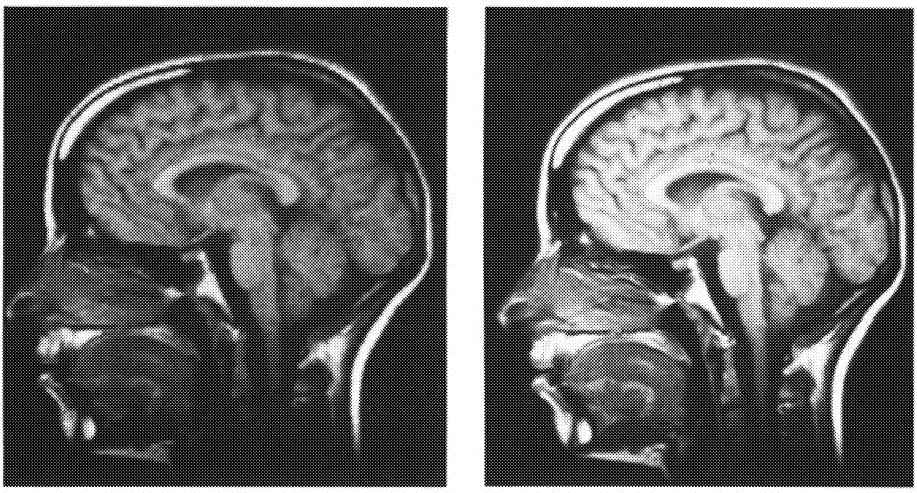


Figure 11.20: VQ MR Image at 1.5 bpp

SNR is more common than PSNR as a quality measure. The SNR for the test MRI image is plotted as a function of bit rate in Figure 11.21.

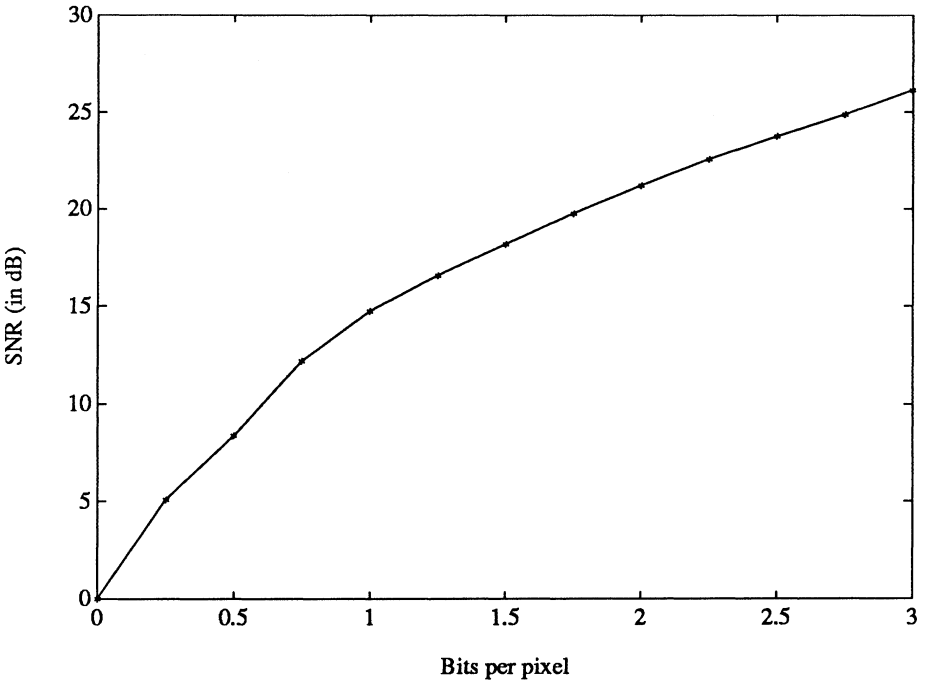


Figure 11.21: PSNR vs. Bits per pixel: Eve

11.5 Problems

11.1. Design a good codebook of size 4 for a Gaussian random vector $\mathbf{X}$ in the plane ($k = 2$). Assume each component of $\mathbf{X}$ has zero mean, unit variance, and the correlation coefficient is 0.9. After computing the codebook, plot the code vectors (indicate with an 'x' or other distinctive sign) on a graph containing the region $-2 \leq X_1 \leq +2$, $-2 \leq X_2 \leq +2$. Plot the first 100 training vectors with a dot ($\cdot$) on the same graph as the code vectors. (Use the generalized Lloyd I algorithm, generating a training set of 1000 random vectors.) Pick any reasonable initial codebook and use a stopping criterion of 1% threshold in relative drop in distortion AND a maximum of 20 Lloyd iterations. Specify the final distortion (averaged over your training set) achieved for your codebook and specify the location of the four code vectors.

11.2. A two-dimensional random vector is uniformly distributed on the unit circle $\{x, y : x^2 + y^2 \leq 1\}$. Find a VQ satisfying the Lloyd conditions for a codebook of size 2. Repeat for codebook sizes 3 and 4. (You are to do this by trial and error rather than by computer, although you can program a solution as a guide. You must prove your quantizers meet the Lloyd conditions.)

Are the codebooks of size 2 and 3 essentially unique, that is, is there only one codebook satisfying these conditions except for rotations and permutations? (Say what you can about this, do not be discouraged if you do not come up with a solution.)

Show that there are clearly distinct codebooks of size 4 by finding one with a vector at the origin and the other without. Which is better?

11.3. A vector quantizer is to be designed to minimize the average weighted distortion measure given by

$$d(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^t \mathbf{W}_{\mathbf{X}} (\mathbf{x} - \mathbf{y})$$

where for each $\mathbf{x} \neq 0$ $\mathbf{W}_{\mathbf{x}}$ is a symmetric positive definite matrix (that depends on $\mathbf{x}$) and $E(\mathbf{W}_{\mathbf{X}})$ and $E(\mathbf{W}_{\mathbf{X}}\mathbf{X})$ are finite. For a given partition, find the necessary condition for the code vectors to be optimal.

11.4. Consider the weighted squared error distortion measure of the previous problem. Find extensions of Lemma 11.2.2 to codes satisfying the centroid condition with respect to this distortion measure. (The results will be different in form, but similar in appearance. If you are stuck, track down [166].)

11.5. A coding system performs some processing on an incoming signal and then generates a random vector $\mathbf{U}$ and a scalar random variable G that is statistically dependent on $\mathbf{U}$. (But G is *not* the gain of $\mathbf{U}$; it depends on other random variables derived from the input signal that are not contained in the vector $\mathbf{U}$.) Specifically, G is typically the rms level of a long segment of the input signal, whereas $\mathbf{U}$ is a vector formed from a small number of consecutive samples of this input segment.) The scalar G provides a rough estimate of the rms level of U (without actually requiring the use of bits to identify the true rms level of $\mathbf{U}$) and is used to normalize $\mathbf{U}$ before applying it to a vector quantizer, $\mathbf{Q}$. The quantizer output is rescaled (multiplied by G) to produce the final estimate $\hat{\mathbf{U}}$ of $\mathbf{U}$. Assume G is transmitted to the receiver with perfect fidelity. Assume that the joint probability distribution of $\mathbf{U}$ and G are known so that all expectations can be determined.

(a) Given the codebook for $\mathbf{Q}$, derive the optimal partition rule for $\mathbf{Q}$.

(b) Given a fixed partition for $\mathbf{Q}$, derive a formula for the optimal codevectors in the codebook for $\mathbf{Q}$ which minimize the average squared Euclidean distance between $\mathbf{U}$ and $\hat{\mathbf{U}}$. *Hint:* one way to do this it to use a natural generalization of the orthogonality condition; alternatively, you can express the desired codevector in terms of its (scalar) components and optimize separately for each component.

11.6. Design and sketch a good 2-dimensional quantizer with 3 levels for an iid sequence of uniform random variables (uniform on [-1,1]) and the mean squared error fidelity criterion using the Lloyd algorithm and a training sequence of at least 5000. What is the final mean squared error? What is the mean squared error if your code is run on a separate test sequence (produced by the same random number generator with a different seed or initial value)?

11.7. Show that the conclusions of Lemma 11.2.1 remain true for any distortion measure of the form $d(\mathbf{x},\mathbf{y}) = \rho(||\mathbf{x}-\mathbf{y}||)$, where $\rho(\cdot)$ is a monotonically nondecreasing function and $||\cdot||$ is the Euclidean distance.

11.8. Use the Lloyd algorithm to design a scalar ($k = 1$) quantizer for an iid Gaussian source for codebooks of size $N = 1, 2, 3, 4, 6, 8$ using a training sequence of 10,000 samples. Compare your results with those reported by Max [235].

11.9. Given that a random vector $\mathbf{X}$ lies in a convex set R, show that its mean value lies in R.

11.10. Consider a convex bounded region R in k-dimensional space with volume V. Assume that the region contains N training vectors that are very roughly uniformly distributed in R and that the nearest neighbor region for each vector is roughly approximated as a hypersphere. Find a formula for the radius of each hypersphere and indicate how this can be used for selecting a threshold value for the pruning method of designing an initial VQ codebook.

11.11. You are given the following 4-dimensional binary training set:

$$T = \{(1,1,1,1),(1,1,1,0),(1,1,1,0),(0,0,0,1),(1,0,0,1),$$
$$(0,0,0,1),(1,0,0,0),(0,0,1,0),(0,0,0,1),(1,1,0,1)\}.$$

Use the generalized Lloyd algorithm with the Hamming distance distortion measure to design a 2-codeword VQ. Use the initial codebook

$$Y_1 = \{(1,1,0,0),(0,0,1,1)\}.$$

Assume in the case of a tie in the distortion between an input vector and the two codewords, that the training vector is assigned to the codeword with the lower index, that is, to the cell R_1. Also assume that in the case of a tie in the centroid computation (a majority rule) that a 0 is chosen.

No computer is necessary.

(a) What is the final codebook you get?

(b) What is the average Hamming distortion between the training sequence and the final codebook?

(Problem courtesy of E. Riskin.)

11.12. Use the splitting algorithm to design a 2-word (rate 1) codebook for the training sequence

$$T = \{(0,\tfrac{1}{4}),(\tfrac{1}{4},\tfrac{1}{4}),(\tfrac{3}{4},\tfrac{1}{4}),(1,\tfrac{1}{4}),(\tfrac{1}{4},\tfrac{1}{2}),$$
$$(\tfrac{3}{4},\tfrac{1}{2}),(0,1),(\tfrac{1}{2},1),(1,1),(\tfrac{1}{4},\tfrac{3}{4})\}.$$

You may assume that the perturbation is done with $\epsilon = 0.01$, that is, split the rate 0 codebook y_0 into y_0 and $y_0 + 0.01$.

(a) What is the initial rate 0 codebook?

(b) What is the final rate 1 codebook?

(c) What is the distortion between the training set and the rate 1 codebook?

(Problem courtesy of E. Riskin.)

11.13. Design a nearly optimal codebook of size $N = 5$ for a Gaussian random vector $\mathbf{X}$ in the plane ($k = 2$). Assume each component of $\mathbf{X}$ has zero mean, unit variance, and the correlation coefficient is 0.95. After computing the codebook, plot the codevectors (indicate with an 'x' or other distinctive sign) on a graph containing the region $-2 \leq X_1 \leq +2$, $-2 \leq X_2 \leq +2$. Plot the first 100 training vectors with a dot ($\cdot$) or other distinctive marker on the same graph as the codevectors. Use the generalized Lloyd I algorithm, generating a training set of 1000 random vectors. [First figure out how to generate independent uniform random variables over (0, 1) using a modulo congruence relation or an existing library routine in C or Fortran, then find a way of generating independent Gaussian variables from the uniform variables (lots of literature available on this) and then use take a linear transformation to produce the desired random variables. See also Problem 2.10.] Pick any reasonable initial codebook and use a stopping criterion of 1% threshold in relative drop in distortion AND a maximum of 20 Lloyd iterations. Specify the final mean squared distortion per vector (averaged over your training set) achieved for your codebook and specify the location of the five codevectors.

11.14. Obtain a training sequence for speech and design VQs for 1 bit per sample and dimension 8 using both squared error distortion and the input weighted squared error distortion defined by

$$d(\mathbf{x}, \hat{\mathbf{x}}) = \frac{||\mathbf{x} - \hat{\mathbf{x}}||^2}{||\mathbf{x}||^2},$$

that is, the short term noise to signal ratio. Compare the resulting subjective quality.

11.15. Suppose that a distortion measure is defined to depend on the sample variance of the input in the following manner. Given a k-dimensional input vector $\mathbf{x} = (x_1, \cdots, x_k)^t$, define the sample mean

$$m(\mathbf{x}) = \frac{1}{k} \sum_{l=1}^{k} x_l$$

and the sample variance

$$\sigma^2(\mathbf{x}) = \frac{1}{k}\sum_{l=1}^{k}(x_l - m(\mathbf{x}))^2.$$

An input vector $\mathbf{x}$ is classed as being in C_1 if $\sigma^2(\mathbf{x}) > \delta$ and as being in class C_2 if $\sigma^2(\mathbf{x}) \leq \delta$. The threshold $\delta > 0$ is fixed. Intuitively, the C_1 vectors are more "active" since they possess a larger sample variance. The distortion measure is now defined as

$$d(\mathbf{x}, \hat{\mathbf{x}}) = \begin{cases} \lambda||\mathbf{x} - \hat{\mathbf{x}}||^2 & \text{if } \mathbf{x} \in C_1 \\ (1-\lambda)||\mathbf{x} - \hat{\mathbf{x}}||^2 & \text{if } \mathbf{x} \in C_2 \end{cases},$$

where $\lambda \in (0, 1)$ is a fixed weight. This distortion measures allows us to weight active input vectors as more (or less) important. Describe the details of the generalized Lloyd algorithm for designing a VQ with respect to this distortion measure.

11.16. The partial distortion theorem (Theorem 6.3.1) for scalar quantizers suggests that in addition to the two known Lloyd conditions for optimality, the optimal vector quantizer should perhaps force all of the partial distortions

$$D_i = E[||\mathbf{X} - Q(\mathbf{X})||^2 | \mathbf{X} \in R_i] \Pr(\mathbf{X} \in R_i)$$

to be equal, regardless of i. Is this true? Explain.