Chapter 10

Vector Quantization I: Structure and Performance

10.1 Introduction

Vector quantization (VQ) is a generalization of scalar quantization to the quantization of a vector, an ordered set of real numbers. The jump from one dimension to multiple dimensions is a major step and allows a wealth of new ideas, concepts, techniques, and applications to arise that often have no counterpart in the simple case of scalar quantization. While scalar quantization is used primarily for analog-to-digital conversion, VQ is used with sophisticated digital signal processing, where in most cases the input signal already has some form of digital representation and the desired output is a compressed version of the original signal. VQ is usually, but not exclusively, used for the purpose of data compression. Nevertheless, there are interesting parallels with scalar quantization and many of the structural models and analytical and design techniques used in VQ are natural generalizations of the scalar case.

A vector can be used to describe almost any type of *pattern*, such as a segment of a speech waveform or of an image, simply by forming a vector of samples from the waveform or image. Another example, of importance in speech processing, arises when a set of parameters (forming a vector) is used to represent the spectral envelope of a speech sound. Vector quantization can be viewed as a form of pattern recognition where an input pattern is "approximated" by one of a predetermined set of standard patterns, or in other language, the input pattern is matched with one of a stored set of templates or codewords. Vector quantization can also be viewed as a front end to a variety of complicated signal processing tasks, including classification and linear transforming. In such applications VQ can be viewed as a complexity reducing technique because the reduction in bits can simplify the subsequent computations, sometimes permitting complicated digital signal processing to be replaced by simple table lookups. Thus VQ is far more than a formal generalization of scalar quantization. In the last few years it has become an important technique in speech recognition as well as in speech and image compression, and its importance and application are growing.

Our treatment of VQ in this book is motivated primarily by its value as a powerful technique for data compression. We hope, however, that the treatment presented here will provide a foundation for applications in pattern recognition as well.

The topics presented in this chapter and the next closely parallel those of Chapters 5 and 6 on scalar quantization. Many of the basic definitions and properties immediately generalize from scalars to vectors, while some do not generalize at all. These similarities and differences will be emphasized in the development.

We first present the basic definition of VQ and the structural properties that are independent of any statistical considerations or distortion measures. The structure and basic ideas for software or hardware implementation of VQ are considered for both the general case and the special case of uniform quantizers based on lattices. Basic complexity considerations are also presented. The presentation here assumes the reader has read or reviewed the basic material on scalar quantization presented in Chapters 5 and 6. We shall occasionally refer to "quantizers" or "quantization" implying the generality of VQ but without specifically attaching the modifier "vector." Of course, scalar quantization is always a special case of VQ and all results can so be specialized. In Chapter 11 we continue the treatment of VQ by focusing on the optimality properties of vector quantizers and their implications for quantizer design. In Chapter 12 we consider special structures for VQ that often help to improve distortion-rate performance when a complexity constraint is placed on the encoding operation.

Basic Definitions

A vector quantizer Q of dimension k and size N is a mapping from a vector (or a "point") in k-dimensional Euclidean space, $\mathcal{R}^k$, into a finite set $\mathcal{C}$ containing N output or reproduction points, called *code vectors* or *codewords*. Thus,

$$Q : \mathcal{R}^k \to \mathcal{C},$$

where $C = (\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_N)$ and $\mathbf{y}_i \in \mathcal{R}^k$ for each $i \in \mathcal{J} \equiv \{1, 2, \cdots, N\}$. The set C is called the *codebook* or the *code* and has size N, meaning it has N distinct elements, each a vector in $\mathcal{R}^k$. The *resolution*, *code rate*, or, simply, *rate* of a vector quantizer is $r = (\log_2 N)/k$, which measures the number of bits per vector component used to represent the input vector and gives an indication of the accuracy or precision that is achievable with a vector quantizer if the codebook is well-designed. It is important to recognize that for a fixed dimension k the resolution is determined by the size N of the codebook and not by the number of bits used to numerically specify the code vectors stored in the codebook. The codebook is typically implemented as a table in a digital memory and the number of bits of precision used to represent each component of each code vector does not affect the resolution or the bit-rate of the vector quantizer; it is of concern only in connection with storage space limitations and with the question of adequate precision in describing a well-designed codebook.

Associated with every N point vector quantizer is a *partition* of $\mathcal{R}^k$ into N regions or *cells*, R_i for $i \in \mathcal{J}$. The ith cell is defined by

$$R_i = \{\mathbf{x} \in \mathcal{R}^k : Q(\mathbf{x}) = \mathbf{y}_i\}, \tag{10.1.1}$$

sometimes called the *inverse image* or *pre-image* of $\mathbf{y}_i$ under the mapping Q and denoted more concisely by $R_i = Q^{-1}(\mathbf{y}_i)$.

From the definition of the cells, it follows that

$$\bigcup_i R_i = \mathcal{R}^k \text{ and } R_i \bigcap R_j = \emptyset \text{ for } i \neq j, \tag{10.1.2}$$

so that the cells form a partition of $\mathcal{R}^k$. A cell that is *unbounded* is called an *overload cell* and the collection of all overload cells is called the *overload region*. A bounded cell, i.e., one having finite (k-dimensional) volume, is called a *granular cell*. The collection of all granular cells is called the granular region.

An important property of a set in $\mathcal{R}^k$ is *convexity*. Recall that in two or three dimensions, a set is said to be convex if given any two points in the set, the straight line joining these two points is also a member of the set. This familiar idea remains applicable in $\mathcal{R}^k$. A set $\mathbf{S} \in \mathcal{R}^k$ is convex if a and $b \in \mathbf{S}$ implies that $\alpha a + (1-\alpha)b \in \mathbf{S}$ for all $0 < \alpha < 1$.

Definition: A vector quantizer is called *regular* if

a) Each cell, R_i, is a convex set, and

b) For each i, $\mathbf{y}_i \in R_i$.

It is also convenient to define a *polytopal vector quantizer* as a regular quantizer whose partition cells are bounded by segments of hyperplane surfaces in k dimensions. Equivalently, each partition region is a regular polytope and consists of an intersection of a finite number of half spaces of the form $\{\mathbf{x} \in \mathcal{R}^k : \mathbf{u}_\nu \cdot \mathbf{x} + \beta_\nu \geq 0\}$. For a thorough treatment of polytopes, the generalization of polyhedra, see Coxeter [82]. The *faces* of a polytopal cell are hyperplane segments of dimension less than k that bound the cell, so that every point on one side of the face is inside the cell and every point on the other side of the face is outside the cell. Usually, a face refers to a $k-1$ dimensional hyperplane segment for a cell in k dimensions. Note that the definition of a regular vector quantizer is consistent with the scalar case and that a regular quantizer in the one-dimensional case is always polytopal.

A vector quantizer is said to be *bounded* if it is defined over a bounded domain, $B \subset \mathcal{R}^k$, so that every input vector, $\mathbf{x}$, lies in this set. The *volume* of the set B, denoted by $V(B)$ and given by

$$V(B) = \int_B d\mathbf{x}, \tag{10.1.3}$$

is therefore finite. A bounded quantizer does not have any overload regions in its partition.

A vector quantizer can be decomposed into two component operations, the vector *encoder* and the vector *decoder*. The encoder $\mathcal{E}$ is the mapping from $\mathcal{R}^k$ to the index set $\mathcal{J}$, and the decoder $\mathcal{D}$ maps the index set $\mathcal{J}$ into the reproduction set $\mathcal{C}$. Thus,

$$\mathcal{E} : \mathcal{R}^k \rightarrow \mathcal{J} \text{ and } \mathcal{D} : \mathcal{J} \rightarrow \mathcal{R}^k. \tag{10.1.4}$$

It is important to note that a given partition of the space into cells fully determines how the encoder will assign an index to a given input vector. On the other hand, a given codebook fully determines how the decoder will generate a decoded output vector from a given index. The task of the encoder is either implicitly or explicitly to identify in which of N geometrically specified regions of k space the input vector lies. Contrary to popular belief, the encoder does not fundamentally need to know the codebook to perform its function. On the other hand, the decoder is simply a table lookup which is simply and fully determined by specifying the codebook. The decoder does not need to know the geometry of the partition to perform its job. Later we shall see that for most vector quantizers of practical interest, the codebook provides sufficient information to characterize the partition and in this case, the encoder operation can be performed by using the codebook as the data set which implicitly specifies the partition.

The overall operation of VQ can be regarded as the cascade or composition of two operations:

$$Q(\mathbf{x}) = \mathcal{D} \cdot \mathcal{E}(\mathbf{x}) = \mathcal{D}(\mathcal{E}(\mathbf{x})). \tag{10.1.5}$$

Occasionally it is convenient to regard a quantizer as generating both an index, i, and a quantized output value, $Q(x)$. The decoder is sometimes referred to as an "inverse quantizer." Figure 10.1 illustrates how the cascade of an encoder and decoder defines a quantizer.

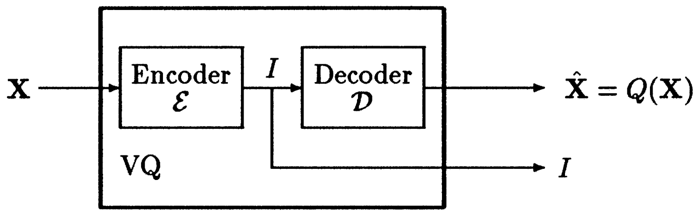

Figure 10.1: A Vector Quantizer as the Cascade of an Encoder and Decoder

In the context of a digital communication system, the encoder of a vector quantizer performs the task of selecting (implicitly or explicitly) an appropriately matching code vector $\mathbf{y}_i$ to approximate, or in some sense to describe or represent, an input vector $\mathbf{x}$. The index i of the selected code vector is transmitted (as a binary word) to the receiver where the decoder performs a table-lookup procedure and generates the reproduction $\mathbf{y}_i$, the quantized approximation of the original input vector. If a sequence of input vectors is to be quantized and transmitted, then the *bit-rate* or *transmission rate* R, in bits per vector, is given by $R = kr$, where r is the resolution and k the vector dimension. If we let f_v denote the *vector rate*, or the number of input vectors to be encoded per second, then the *bit-rate*, R_s, in bits per second is given by $R_s = krf_v$.

Of particular interest is the case of a scalar waveform communication system where each vector represents a block of contiguous samples of the waveform. The vector sequence, called a *blocked scalar process*, then corresponds to consecutive blocks of the waveform and the vector dimension, k, and the sampling rate, f_s, measured in samples per second, together determine the vector rate, in this case $f_v = f_s/k$ vectors per second. The bit-rate in bits per second is therefore, $R_s = rf_s$ which is independent of the dimension k. We distinguish between resolution and rate since there are important applications of VQ where the vectors are extracted parameters of a signal rather than blocked samples of a waveform.

Generality of Vector Quantization

VQ is not merely a generalization of scalar quantization. In fact, it is the "ultimate" solution to the quantization of a signal vector. No other coding technique exists that can do better than VQ. This appears to be a sweeping statement that may leave the reader somewhat skeptical at first. Yet it is readily justified. The following theorem shows that VQ can at least match the performance of any arbitrarily given coding system that operates on a vector of signal samples or parameters.

Theorem 10.1.1 *For any given coding system that maps a signal vector into one of N binary words and reconstructs the approximate vector from the binary word, there exists a vector quantizer with codebook size N that gives exactly the same performance, i.e., for any input vector it produces the same reproduction as the given coding system.*

Proof: Enumerate the set of binary words produced by the coding system as indexes $1, 2, \ldots, N$. For the ith binary word, let the decoded output of the given coding system be the vector $\mathbf{y}_i$. Define the codebook $\mathcal{C}$ as the ordered set of code vectors $\mathbf{y}_i$. Then a VQ decoder achieves equivalent performance to the decoder of the given coding system and a VQ encoder can be defined to be identical to the encoder of the given coding system. □

In general, an *ad hoc* or heuristically designed coding technique that codes a set of k signal samples or parameters with b bits, is a suboptimal way to map a k dimensional input vector into one of $N = 2^b$ index values and into one of N reproduction vectors. Since VQ encompasses all such possible schemes, it is natural to seek the optimal encoding scheme by starting with the most general way of modeling the encoding problem, namely VQ. In the chapter that follows we consider this problem, that is, the joint optimization of the encoder and decoder for VQ with a given performance measure and given input statistics. At this point, it should be clear that if we can find the optimal vector quantizer for a given performance objective, no other coding system will be able to achieve a greater performance.

Examples of Vector Quantization

It is convenient to view the operation of a vector quantizer geometrically, using our intuition for the case of two- or three-dimensional space. Thus, a 2-dimensional quantizer assigns any input point in the plane to one of a particular set of N points or locations in the plane. As a simple illustration, consider a map of a city that is divided into school districts and the codebook is simply the location of each school on the map. The "input" is the location of a particular child's residence and the quantizer is simply the rule that assigns each child to a school according to the child's location.

Figure 10.2 demonstrates an example of a two-dimensional quantizer that is neither polytopal nor regular since the cells have faces (one-dimensional boundaries) that are not segments of hyperplanes (straight line segments) and the cells are not convex. The dots represent code vectors in a

2 dimensional space (the plane) and the region containing each codevector is a partition cell. Figure 10.3 shows a two-dimensional regular quantizer

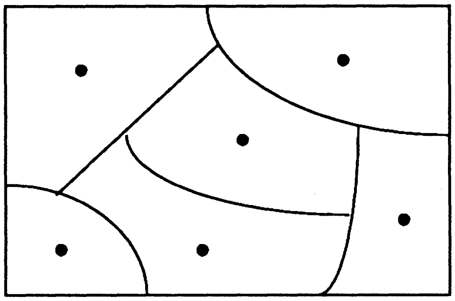

Figure 10.2: A Nonregular Quantizer

whose bounded cells are polygons (closed polytopes in two dimensions).

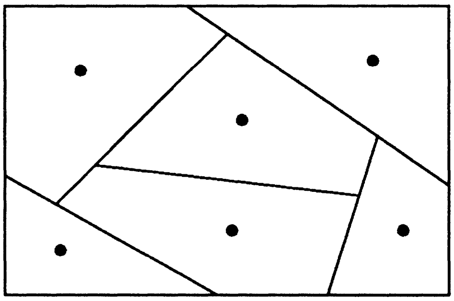

Figure 10.3: A Regular Quantizer

As an illustration of the generality of VQ, consider the coding scheme where two random variables x_1 and x_2 are each quantized by a scalar quantizer. As indicated by Theorem 10.1.1 on the generality of VQ, this case can be considered as a degenerate special case of vector quantization of the vector $\mathbf{x} = (x_1, x_2)$ where the vector quantizer is given by

$$Q(\mathbf{x}) = (Q_1(x_1), Q_2(x_2)) \tag{10.1.6}$$

where Q_1 and Q_2 are the scalar quantizers for x_1 and x_2, respectively. Figure 10.4 shows the resulting vector quantizer corresponding to a particular choice of scalar quantization for each variable. We note in passing that this is an example of a *product VQ* because the overall VQ is formed as a Cartesian product of smaller dimensional VQs. Product codes are studied in more detail in Chapter 12. It is evident that the VQ defined by separately

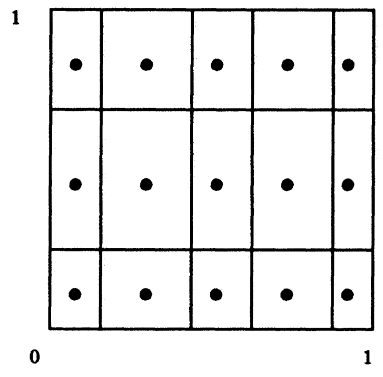

Figure 10.4: VQ Based Upon Scalar Quantization

quantizing the components of a vector must *always* result in quantization cells that are rectangular. In contrast, a more general vector quantizer is freed from these geometrical restrictions and can have arbitrary cell shapes as indicated in the examples of Figures 10.2 and 10.3. In higher dimensions the same idea is clearly applicable. Thus, in three dimensions, scalar quantization of the three components of a vector always results in cells that have rectangular box-like shapes where each face is a plane parallel to one of the coordinate axes. On the other hand, regular quantizers in three dimensions will have polyhedral cells. (A polyhedron is a polytope in three dimensions.) Extending this idea to k dimensions, it is clear that scalar quantization of the components of a vector always generates a very restricted class of vector quantizers where the faces are $(k-1)$-dimensional hyperplanes each parallel to a coordinate axis in the k-dimensional space. The inherent superiority of VQ is thereby evident simply because of the greater structural freedom it allows in quantization of a vector.

As another example of a special but restricted case of VQ, consider transform coding of a k dimensional vector as treated in Chapter 8. Since transform coding with an orthogonal matrix is equivalent to a rotation of the coordinate system, the use of scalar quantization on the transform coefficients is equivalent to a partition of the input vector space into rectangular cells which are parallel to the axes of a rotated coordinate system. From Theorem 10.1.1, transform coding can be considered as a special case of VQ whose performance is limited by the imposed coding structure. While general VQ is based on a partition of the input space into N arbitrarily shaped regions, the partition cells for transform coding are restricted to shapes that are rotations of those induced by scalar coding.

10.2 Structural Properties and Characterization

A structural decomposition of a vector quantizer is particularly valuable for obtaining insight needed to find effective algorithms for implementing vector quantizers [138]. It is also valuable for analytical studies including vector quantizer optimization. As in the scalar case, it is desirable to find a decomposition into simple building blocks and to identify the primitive building blocks that can be used to construct a model of a vector quantizer. We begin with the primary structural decomposition which is a direct generalization of the scalar case. Then we present the secondary structure, a nontrivial and important decomposition of a vector quantizer that reveals the intriguing richness of the multidimensional case. Following naturally after this background is an examination of the encoding complexity and encoding techniques for VQ. An important special class of vector quantizers, lattice-based quantizers, is treated next.

Primary Structure of a Vector Quantizer

The encoding task of a quantizer is to examine each input vector $\mathbf{x}$ and identify in which partition cell of the k-dimensional space $\mathcal{R}^k$ it lies. The vector encoder simply identifies the index i of this region and the vector decoder generates the code vector $\mathbf{y}_i$ that represents this region.

To model the operation of the encoder, we define as in the scalar case the *selector function*, $S_i(\mathbf{x})$, as the indicator or membership function $1_{R_i}(\mathbf{x})$ for the cell R_i of the partition, that is,

$$S_i(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in R_i \\ 0 & \text{otherwise.} \end{cases} \qquad (10.2.1)$$

The operation of a vector quantizer can then be represented as:

$$Q(\mathbf{x}) = \sum_{i=1}^{N} \mathbf{y}_i S_i(\mathbf{x}). \qquad (10.2.2)$$

Note that for any given value of the input vector $\mathbf{x}$, only one term of the sum is nonzero.

Figure 10.5 shows this model of a vector quantizer, which we call the *primary structural decomposition*. Each multiplier, indicated by circles, simply multiplies a particular code vector by either one or zero to produce as its output either the code vector or the zero vector. Alternatively, the multiplier can be viewed as a memory cell that stores a code vector whose value is retrieved when the binary input has value 1. The summation of

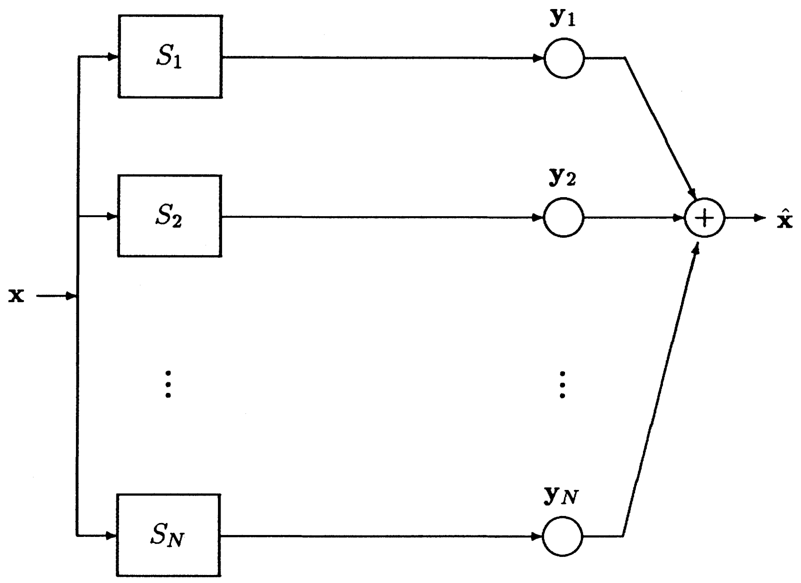

Figure 10.5: Primary Structural Decomposition of VQ

course is only symbolic since only one of its inputs is nonzero. Each S box is a memoryless (in terms of vectors) nonlinear operation which examines all components of the input vector simultaneously in order to evaluate whether or not the input belongs to the particular cell. The task of computing the binary output of an S box can indeed be a rather complex operation depending on the particular geometrical character of the cell in k-dimensional Euclidean space. The internal operation of the S box, however, is not of explicit interest in the primary decomposition. It is often convenient to abbreviate $S_i(\mathbf{x})$ to simply S_i and consider it the binary valued output of the ith selector box.

As in the scalar case, the encoder and decoder can be separately modeled with a corresponding primary structure. Let A denote the *index generator*, as the mapping $A : \mathcal{B}_N \rightarrow \mathcal{J}$ and its inverse, the *index decoder*, $A^{-1} : \mathcal{J} \rightarrow \mathcal{B}_N$, where $\mathcal{B}_N$ denotes the set of binary N-tuples of the form $\mathbf{b} = (b_1, b_2, \cdots, b_N)$ with $b_i \in \{0,1\}$ and $\mathbf{b}$ is restricted to have exactly one nonzero component. Then $A(\mathbf{b}) = i$ if $b_i = 1$ and $b_j = 0$ for $j \neq i$. Then the encoder operation can be written as

$$\mathcal{E}(\mathbf{x}) = A(S_1(\mathbf{x}), S_2(\mathbf{x}), \cdots, S_N(\mathbf{x})) \tag{10.2.3}$$

and the corresponding decoder operation is given by

$$\mathcal{D}(i) = \sum_{l=1}^{N} \mathbf{y}_l A^{-1}(i)_l. \qquad (10.2.4)$$

The structural diagrams corresponding to the encoder and decoder are shown in Figure 10.6. In the context of a communications system, the

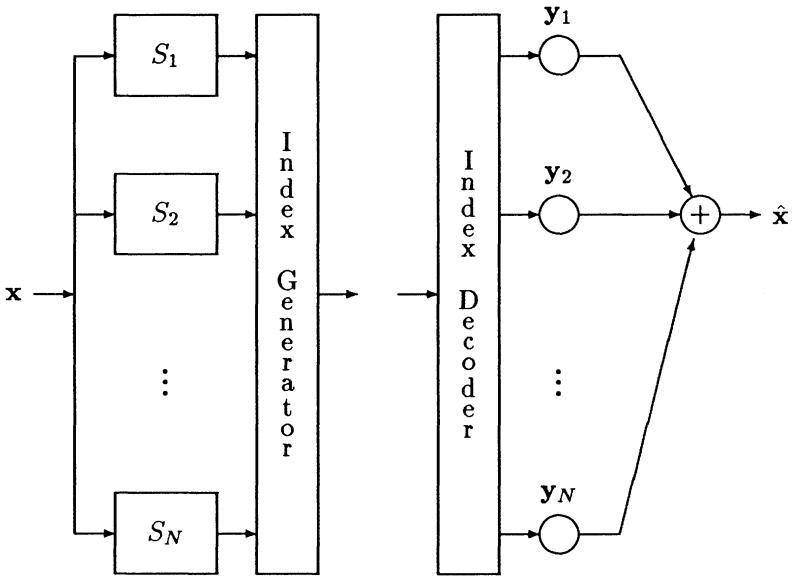

Figure 10.6: Primary Structure of a Vector Encoder and Vector Decoder

index i may be regarded as the identifier of a channel symbol whose physical nature can have many forms. The index generator and index decoder correspond respectively to the address encoder and address decoder as used in digital memory circuits.

Secondary Structure of Polytopal Vector Quantizers

The selector function for a particular cell of the vector quantizer can be a rather complex operation and it does not fit with the notion of a primitive building block for modeling a quantizer. Further decomposition of the selector function is indeed feasible, but it is dependent on the particular classification of the vector quantizer. Since the cells that partition $\mathcal{R}^k$ for a vector quantizer can have arbitrarily intricate geometric shapes, there are no natural "elementary" geometrical components that can be used as universal building blocks to model the selector operation. We now focus on a restricted but extremely important and widely used class of quantizers, the polytopal quantizers, in order to obtain a secondary decomposition of the structure of VQ.

We have defined a regular vector quantizer as ***polytopal*** if each cell in the partition is a convex polytope, that is, the $(k-1)$-dimensional faces of the cells consist of hyperplane segments. To be more explicit, consider the ***half-space*** denoted by

$$H_\nu = \{\mathbf{x} \in \mathcal{R}^k : \mathbf{u}_\nu \cdot \mathbf{x} + \beta_\nu \geq 0\}, \tag{10.2.5}$$

where the dot product denotes the usual scalar product or inner product

$$\mathbf{u} \cdot \mathbf{x} = \mathbf{u}^t\mathbf{x} = \sum_{i=0}^{k-1} u_i x_i$$

between two vectors. The half-space H_ν is characterized by the vector parameter $\mathbf{u}_\nu \in \mathcal{R}^k$ and the scalar value β_ν. Let

$$\Lambda_\nu = \{\mathbf{x} \in \mathcal{R}^k : \mathbf{u}_\nu \cdot \mathbf{x} + \beta_\nu = 0\}, \tag{10.2.6}$$

be the hyperplane that bounds H_ν. Then, any polytopal region can be represented by a finite intersection of half planes and the associated hyperplanes contain the "faces" or boundaries of the polytope. In particular, for any polytopal vector quantizer the partition cells can be written as

$$R_i = \bigcap_{\nu=1}^{L_i} H_\nu, \tag{10.2.7}$$

where L_i is equal to the number of $(k-1)$-dimensional faces of the cell R_i. Let $u(v)$ denote the unit step function, where $u(v) = 1$ for $v \geq 0$ and $u(v) = 0$ for $v < 0$. Then the indicator function $T_\nu(\mathbf{x}) = 1_{H_\nu}(\mathbf{x})$ for the half-space H_ν is defined by

$$T_\nu(\mathbf{x}) = u(\mathbf{u}_\nu \cdot \mathbf{x} + \beta_\nu). \tag{10.2.8}$$

Finally, we obtain

$$S_i(\mathbf{x}) = \prod_{\nu=1}^{L_i} T_\nu(\mathbf{x}), \tag{10.2.9}$$

an explicit decomposition of the cell selector function into a product of binary-valued hyperplane decision functions. Observe that we could replace the product by a logical AND; the key idea is that S_i is 1 if and only if all of the T_ν are 1. A hyperplane decision function, simply answers the question:

"Is $\mathbf{x}$ on the positive side of hyperplane Λ_ν?"

It is quite reasonable to consider this as a primitive operation in k-dimensional space. Figure 10.7 illustrates this decomposition of the selector function into a product of such primitive hyperplane decision functions.

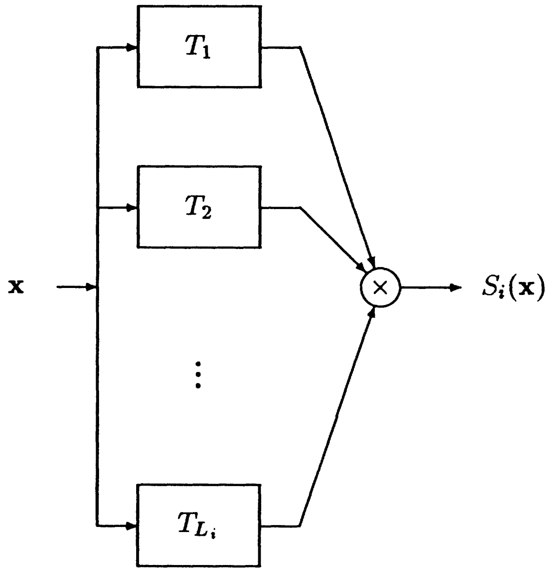

Figure 10.7: Structural Decomposition of a Selector Function

It should be noted that the complexity of the VQ encoder can be very large since the number of hyperplanes that typically bound a polytope grows rapidly with the dimension of the space.

The above representation assumed that the cells are convex as well as polytopal. This condition is indeed apparent from the decomposition since any half-space as defined above is readily shown to be convex and each cell is the intersection of convex sets and is thereby convex.

The primitive elements of this structure, which we have called hyperplane decision functions, can also be recognized as fundamental elements in the field of pattern recognition where they are typically known as *linear decision functions*. These elements are closely related to perceptrons and arise in neural network modeling. In the one-dimensional case, the hyperplane decision functions reduce to the binary threshold elements discussed in Chapter 5.

Tertiary Structure of a Polytopal Vector Quantizer

If indeed the hyperplane decision functions are appropriately considered as primitive elements, the secondary structure of a polytopal quantizer offers the last word in decomposition into elementary building blocks. However, there is an alternative approach that leads to a different structure which has in some sense greater simplicity, at least on a conceptual level. By further decomposing the hyperplane decision function, we obtain a *tertiary decomposition* of a polytopal vector quantizer.

Consider one particular selector function, say $S_i(\mathbf{x})$. Form the $L_i \times k$ matrix $\mathbf{M}_i$ whose νth row is the vector $\mathbf{u}_\nu$ as defined for the secondary structure. Then the linear transformation

$$\mathbf{w} = \mathbf{M}_i\mathbf{x} \tag{10.2.10}$$

maps the k dimensional input vector $\mathbf{x}$ into the L_i-dimensional transformed vector $\mathbf{w}$. Define the scalar encoders,

$$\mathcal{E}_\nu(w_\nu) = u(w_\nu + \beta_\nu), \tag{10.2.11}$$

where w_ν is the νth component of the transformed vector $\mathbf{w}$. Observe that if the original input is $\mathbf{x}$, then $\mathcal{E}_\nu(w_\nu)$ is just $T_\nu(\mathbf{x})$, the indicator function for the half plane H_ν. The one-bit outputs of these scalar encoders can be applied to an AND gate whose output is a logical one if and only if $\mathbf{x}$ lies in the regions S_j. An input vector can therefore be encoded by performing N linear transformations on the input vector $\mathbf{x}$ followed by one-bit scalar encoding of the transformed vector components. Finally, some combinational logic operations will produce the binary representation of the correct index.

The set of N linear transformations can be combined into one higher dimensional linear transformation matrix whose order is $L \times k$ where $L = \sum_{j=1}^{N} L_j$. Then by performing one-dimensional scalar encodings of all the transformed variables, followed by combinational logic operations the encoder can be implemented with the general structure is illustrated in Figure 10.8, where the comparator inputs are the transformed input vector coordinates and $b_k = -\beta_k$. Therefore, *any polytopal vector encoder can be decomposed into a linear transformation followed by scalar encodings and logical operations.* It is interesting to note the similarity with the operation of transform coding as described in Chapter 8. Although transform coding leads to a very restricted structure of the partition, nevertheless any vector quantizer can be modeled in a manner similar to transform coding. The key distinction here, is that the size of the transform matrix needed to model a given vector quantizer can be much larger than the vector dimension.

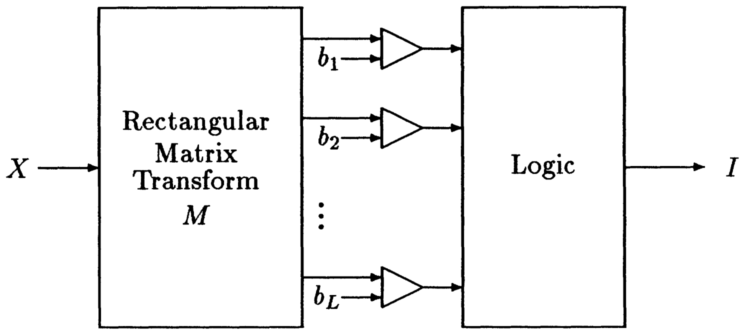

Figure 10.8: Tertiary Structural Decomposition of a VQ based on a Linear Transformation, Scalar Encoders, and Logical Operations

10.3 Measuring Vector Quantizer Performance

A distortion measure d is an assignment of a nonnegative cost $d(\mathbf{x},\hat{\mathbf{x}})$ associated with quantizing any input vector $\mathbf{x}$ with a reproduction vector $\hat{\mathbf{x}}$. Given such a measure we can quantify the performance of a system by an average distortion $D = Ed(\mathbf{X},\hat{\mathbf{X}})$ between the input and the final reproduction. Generally, the performance of a compression system will be good if the average distortion is small.

In practice, the overall measure of performance is the long term sample average or time average

$$\bar{d} = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} d(\mathbf{X}_i,\hat{\mathbf{X}}_i), \tag{10.3.1}$$

where $\{\mathbf{X}_\nu\}$ is a sequence of vectors to be encoded. If the vector process is stationary and ergodic, then with probability one the above limit exists and equals the statistical expectation, i.e., $\bar{d} = D$. Stationarity and ergodicity, however, are not necessary and similar properties can hold under more general conditions. (See, for example, Chapter 1 of [159].) For the moment we will assume that such conditions are met and that such long term sample averages are given by expectations. Later remarks will consider the general assumptions required and their implications for practice.

Let $\mathbf{X}$ denote a continuously distributed random vector in $\mathcal{R}^k$ with a specified pdf (in this case the pdf refers to a joint probability density function of the vector components, X_i, for $i = 1, 2, \cdots, k$). Then the average distortion can be expressed as

$$D = Ed(\mathbf{X}, \mathbf{Y}) = \int_{\mathcal{R}^k} d(\mathbf{x}, Q(\mathbf{x})) f_{\mathbf{X}}(\mathbf{x})\, d\mathbf{x} \tag{10.3.2}$$

and using the partition and codebook for the given quantizer Q, we get

$$D = \sum_{j=1}^{N} \int_{R_j} d(\mathbf{x}, \mathbf{y}_j) f_{\mathbf{X}}(\mathbf{x})\, d\mathbf{x}, \tag{10.3.3}$$

or,

$$\begin{aligned} D &= \int \sum_{j=1}^{N} P_j d(\mathbf{x}, \mathbf{y}_j) f_{\mathbf{X}|j}(\mathbf{x})\, d\mathbf{x}\,, \\ &= \sum_{j=1}^{N} P_j E[d(\mathbf{x}, \mathbf{y}_j) | \mathbf{X} \in R_j], \end{aligned} \tag{10.3.4}$$

where $P_j = P(\mathbf{X} \in R_j)$, and $f_{\mathbf{X}|j}(\mathbf{x}|j)$ is the conditional probability density of $\mathbf{X}$ given that $\mathbf{X} \in R_j$. (Note that $f_{\mathbf{X}|j}(\mathbf{x}) = 0$ for $\mathbf{x} \notin R_j$, so that the integral in (10.3.4) is taken over the entire space $\mathcal{R}^k$). Usually we omit any indication of the range of integration when the integral is taken over the entire k dimensional space.

Occasionally, it is preferable to use a *worst-case* distortion as a measure of performance rather than an average value. With respect to an arbitrary measure $d(\mathbf{x}, \mathbf{y})$ of distortion, this is simply defined as the maximum attainable distortion for a given quantizer:

$$D_{\max} = \max_{\mathbf{x} \in B} d(\mathbf{x}, Q(\mathbf{x})) \tag{10.3.5}$$

where B is the closed subset of $\mathcal{R}^k$ to which the input $\mathbf{X}$ is confined, that is, B is the domain over which the pdf of $\mathbf{X}$ is nonzero. In some cases, this maximum may not exist since the distortion can become arbitrarily large and this measure is then not meaningful. Use of the maximum distortion as a performance measure is limited to the case of bounded random vectors or quantizers with countably infinite codebook sizes with no overload regions. For the most part, we shall focus on the statistical average of the distortion as a performance measure.

Particular Distortion Measures of Interest

Ideally a distortion measure should be tractable to permit analysis and design and it should be computable so that it can be evaluated in real time for guiding the actual encoding process for encoders which select a nearest neighbor or minimum distortion output (we shall see that as with scalar quantizers, this form of encoder is optimal for a given codebook). It should also be subjectively meaningful so that large or small average distortion values correlate with bad and good subjective quality as perceived by the ultimate user of the reproduced vector sequence.

The most convenient and widely used measure of distortion between an input vector $\mathbf{X}$ and a quantized vector $\hat{\mathbf{X}} = Q(\mathbf{X})$, is the *squared error* or squared Euclidean distance between two vectors defined as

$$\begin{aligned} d(\mathbf{X},\hat{\mathbf{X}}) &= \|\mathbf{X}-\hat{\mathbf{X}}\|^2 \\ &\equiv (\mathbf{X}-\hat{\mathbf{X}})^t(\mathbf{X}-\hat{\mathbf{X}}) \\ &= \sum_{i=1}^{k}(X_i-\hat{X}_i)^2. \end{aligned} \tag{10.3.6}$$

If the input and reproduction vectors are complex, then this becomes

$$\begin{aligned} d(\mathbf{X},\hat{\mathbf{X}}) &= \|\mathbf{X}-\hat{\mathbf{X}}\|^2 \\ &\equiv (\mathbf{X}-\hat{\mathbf{X}})^*(\mathbf{X}-\hat{\mathbf{X}}) \\ &= \sum_{i=1}^{k}|X_i-\hat{X}_i|^2. \end{aligned} \tag{10.3.7}$$

The *average squared error distortion* or, more briefly, the *average distortion* (when no confusion with other distortion measures arises) is defined as

$$D = Ed(\mathbf{X},\hat{\mathbf{X}}) = E(\|\mathbf{X}-\hat{\mathbf{X}}\|^2). \tag{10.3.8}$$

This measure is frequently associated with the energy or power of an error signal and therefore has some intuitive appeal in addition to being an analytically tractable measure for many purposes.

Numerous alternative distortion measures may also be defined for assessing dissimilarity between the input and reproduction vectors. (See, for example, Chapter 2 of [159].) Many of the measures of interest for VQ have the form

$$d(\mathbf{X},\hat{\mathbf{X}}) = \sum_{i=1}^{k} d_m(X_i,\hat{X}_i) \tag{10.3.9}$$

where $d_m(x,\hat{x})$ is a scalar distortion measure (often called the *per-letter distortion*) as in one-dimensional quantization. Any distortion measure having this additivity property with the same scalar distortion measure used for each component is called an *additive* or *single letter* distortion measure and is particularly appropriate for waveform coding where each vector component has the same physical meaning, being a sample of a waveform.

Of particular interest is the case where the scalar distortion measure is given by $d_m(x,\hat{x}) = |x-\hat{x}|^m$ for positive integer values of m. When $m = 1$, this specializes to the l_1 *norm* of the error vector, $\mathbf{X}-\hat{\mathbf{X}}$. When $m = 2$, we obtain the squared error measure already discussed. The mth root of d_m is the l_m norm of the error vector $\mathbf{X}-\hat{\mathbf{X}}$.

Another distortion measure of particular interest is the ***weighted squared error*** measure

$$d(\mathbf{x},\mathbf{y}) = (\mathbf{x}-\mathbf{y})^t\mathbf{W}(\mathbf{x}-\mathbf{y}), \tag{10.3.10}$$

where $\mathbf{W}$ is a symmetric and positive definite weighting matrix and the vectors $\mathbf{x}$ and $\mathbf{y}$ are treated as column vectors. Note that this measure includes the usual squared error distortion in the special case where $\mathbf{W} = \mathbf{I}$, the identity matrix. In the case where $\mathbf{W}$ is a diagonal matrix with diagonal values $w_{ii} > 0$, we have

$$d(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{k} w_{ii}(x_i - y_i)^2 \tag{10.3.11}$$

which is a simple but useful modification of the squared error distortion that allows a different emphasis to be given to different vector components.

It should be noted that the weighted squared error measure (10.3.10) can be viewed as an unweighted squared error measure between linearly transformed vectors, $\mathbf{x}' = \mathbf{A}\mathbf{x}$, and $\mathbf{y}' = \mathbf{A}\mathbf{y}$ where $\mathbf{A}$ is obtained by the factorization $\mathbf{W} = \mathbf{A}^t\mathbf{A}$. In some applications, the matrix $\mathbf{W}$ is selected in accordance with statistical characteristics of the input vector, $\mathbf{X}$, being quantized. For example, in the ***Mahalanobis distortion measure***, $\mathbf{W}$ is chosen to be the inverse of the covariance matrix of the input vector $\mathbf{X}$. In particular, if the components of $\mathbf{X}$ are uncorrelated, the Mahalanobis weighting matrix reduces to the diagonal matrix discussed above with $w_{ii} = \sigma_i^{-2}$, where σ_i^2 is the variance of X_i.

All of the distortion measures discussed so far are symmetric in their arguments $\mathbf{x}$ and $\mathbf{y}$. It is sometimes convenient and effective to choose a weighting matrix $\mathbf{W}(\mathbf{x})$ (assumed to be symmetric and positive definite for all $\mathbf{x}$) that depends explicitly on the input vector $\mathbf{x}$ to be quantized in order to obtain perceptually motivated distortion measures for both speech and image compression. In this case, the distortion

$$d(\mathbf{x},\mathbf{y}) = (\mathbf{x}-\mathbf{y})^t\mathbf{W}(\mathbf{x})(\mathbf{x}-\mathbf{y}) \tag{10.3.12}$$

is in general asymmetric in $\mathbf{x}$ and $\mathbf{y}$.. As an example of such a distortion measure, let $\mathbf{W}(\mathbf{x})$ be $||\mathbf{x}||^{-2}\mathbf{I}$, where $\mathbf{I}$ is the identity matrix. Here the distortion between two vectors is the noise energy to signal energy ratio:

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

This allows one to weight the distortion as being more important when the signal is small than when it is large. Note the distortion measure is not well defined unless $||\mathbf{x}|| > \mathbf{0}$.

Finally, we define the *maximum* or l_∞ *norm* distortion measure, by:

$$d_{\max}(\mathbf{x}, \hat{\mathbf{x}}) = \max_i |x_i - \hat{x}_i| \tag{10.3.13}$$

where the distortion is determined by the component of the error vector $\mathbf{x} - \hat{\mathbf{x}}$ that contributes the largest absolute error. It is a well-known mathematical result that the l_m norm approaches the l_∞ norm as $m \to \infty$. Thus,

$$\lim_{m \to \infty} [d_m(\mathbf{x}, \hat{\mathbf{x}})]^{1/m} = d_{\max}(\mathbf{x}, \hat{\mathbf{x}}). \tag{10.3.14}$$

10.4 Nearest Neighbor Quantizers

An important special class of vector quantizers that is of particular interest, called *Voronoi* or *nearest neighbor* vector quantizers, has the feature that the partition is completely determined by the codebook and a distortion measure. We shall see in the next chapter that for a given codebook such an encoder is in fact optimal in the sense of minimizing average distortion, as was the case for scalar quantization. In fact, the term "vector quantizer" is commonly assumed to be synonymous with "nearest neighbor vector quantizer." An advantage of such an encoder is that the encoding process does not require any explicit storage of the geometrical description of the cells. Instead, a conceptually simple algorithm can encode by referring to the stored codebook.

Suppose that $d(\mathbf{x}, \mathbf{y})$ is a distortion measure on the input/output vector space, for example the ubiquitous *squared error distortion measure* defined by the squared Euclidean distance between the two vectors:

$$d(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||^2 = \sum_{i=1}^{k} (x_i - y_i)^2. \tag{10.4.1}$$

This is the simplest measure and the most common for waveform coding. While not subjectively meaningful in many cases, generalizations permitting input-dependent weightings have proved useful and often only slightly more complicated.

We define a *Voronoi* or *nearest neighbor* (NN) vector quantizer as one whose partition cells are given by

$$R_i = \{\mathbf{x} : d(\mathbf{x}, \mathbf{y}_i) \le d(\mathbf{x}, \mathbf{y}_j) \text{ all } j \in \mathcal{J}\}. \tag{10.4.2}$$

In other words, with a nearest neighbor (NN) encoder, each cell R_i consists of all points $\mathbf{x}$ which have less distortion when reproduced with code vector $\mathbf{y}_i$ than with any other code vector. To avoid excessive mathematical formality we have been a bit careless in defining the cells. In order for the cells to constitute a partition, each boundary point must be uniquely assigned to one cell. This modification of the above definition is readily handled by assigning $\mathbf{x}$ to be a member of R_m where m is the smallest index i for which $d(\mathbf{x}, \mathbf{y}_i)$ attains its minimum value.

The most direct encoding algorithm for a NN encoder is given by the following simple algorithm.

Nearest Neighbor Encoding Rule

```
Step 1. Set $d = d_0$, $j = 1$, and $i = 1$.
Step 2. Compute $D_j = d(\mathbf{x}, \mathbf{y}_j)$.
Step 3. If $D_j < d$, set $D_j \rightarrow d$. Set $j \rightarrow i$.
Step 4. If $j < N$, set $j + 1 \rightarrow j$ and go to step 2.
Step 5. Stop. Result is index $i$.
```

The resulting value of i gives the encoder output $\mathcal{C}(\mathbf{x})$ and the final value of d is the distortion between $\mathbf{x}$ and $\mathbf{y}_i$. The initial value, d_0, must be larger than any expected distortion value and is usually set to the largest positive number that can be represented with the processor's arithmetic.

The key feature of the above encoding algorithm is that no geometrical description of the partition is needed to perform the encoding rule. Thus the encoder operation may be described by x

$$\mathcal{E}(\mathbf{x}) = c(\mathbf{x}, \mathcal{C}) \tag{10.4.3}$$

where the functional operation described by $c(\cdot,\cdot)$ is independent of the specific quantizer and depends only on the distortion measure. This important concept is central to the implementation of virtually all vector quantizers in use today. Figure 10.9 illustrates the form of the NN encoder where the

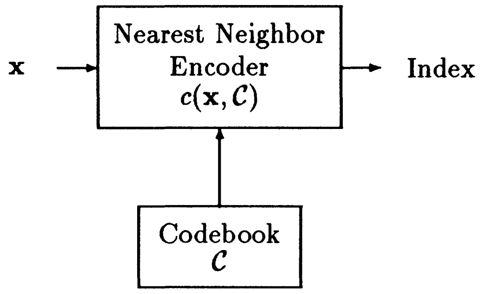

Figure 10.9: Nearest Neighbor Encoder with a Codebook ROM

codebook $\mathcal{C}$ is a read-only memory that is accessed by the NN encoder box that implements the function $c(\cdot, \cdot)$. Virtually any computable distortion measure can be used for a NN encoder. Later, however, we shall see that other properties of a distortion measure are required to yield a statistically based performance measure that is amenable to optimal codebook design.

In general, the NN cells defined above need not be polytopal or even convex. In the important special case where the distortion measure is the squared error measure, however, we shall see that the cells are indeed polytopal so that the secondary and tertiary decompositions described in Section 10.2 are applicable. Furthermore, the NN encoder has the additional feature that the parameters defining the hyperplane decision functions that characterize the cells can be conveniently specified from the values of the code vectors.

Applying (10.4.1) to (10.4.2), we find that the NN cells can be expressed as:

$$R_i = \bigcap_{\substack{j=1 \\ j \neq i}}^{N} H_{ij} \tag{10.4.4}$$

where

$$H_{ij} = \{\mathbf{x} : ||\mathbf{x} - \mathbf{y}_i||^2 \leq ||\mathbf{x} - \mathbf{y}_j||^2\}. \tag{10.4.5}$$

Expanding the squared norms and simplifying, we get

$$H_{ij} = \{\mathbf{x} : \mathbf{u}_{ij} \cdot \mathbf{x} + \beta_{ij} \geq 0\} \tag{10.4.6}$$

where

$$\mathbf{u}_{ij} = 2(\mathbf{y}_i - \mathbf{y}_j), \tag{10.4.7}$$

and

$$\beta_{ij} = ||\mathbf{y}_j||^2 - ||\mathbf{y}_i||^2. \tag{10.4.8}$$

Thus, we see that the NN cells of a Voronoi quantizer are formed by the intersection of half-spaces so that every Voronoi quantizer is polytopal and its half-spaces are explicitly determined from the code vectors $\mathbf{y}_i$ of its codebook. This convenient property is a direct consequence of the fact that the squared error distortion measure is a quadratic function of the code vector components. Other more complex distortion measures which depend on higher order powers of the input vector components will not give rise to polytopal NN quantizers.

For the squared error distortion measure or squared Euclidean distance, the partition cells of the NN encoder not only are polytopal but have an explicitly determined specification derived from the code vectors. This allows a simplification of the general tertiary structure.

Every hyperplane that determines a face of a cell is characterized by two code vectors to which this hyperplane is equidistant. (The converse, on the other hand, is not true: it is not necessary for every pair of code vectors to define a hyperplane that is a face of a cell.) We see that the hyperplane parameter vectors $\mathbf{u}_\nu$ that determine the rows of the transform matrix are given by (10.4.7) for NN cells. Consequently, the only vector-based operation that needs to be performed on the input vector is a linear transform with an $N \times k$ matrix $\mathbf{Y}$ whose rows are the code vectors themselves! In this way, a new N-dimensional vector $\mathbf{a}$ is generated, given by $\mathbf{a} = \mathbf{Y}\mathbf{x}$. Subsequent processing requires selected pairwise subtractions to generate scalar variables of the form $w_{ij} = a_i - a_j$. Finally these values are applied to a one-bit encoder as previously described for the tertiary decomposition. What have we achieved by doing this? The main distinction here is that the order of the linear transformation has been reduced and the matrix itself has an immediate direct interpretation: it is simply a representation of the codebook $\mathcal{C}$ itself.

Finally, we point out a simple and important variation of the nearest neighbor encoding algorithm described earlier. It follows either from the preceding discussion or directly from the squared Euclidean distance measure that the nearest neighbor search can be performed by evaluating scalar products rather than taking squared norms. Since

$$\min_i{}^{-1} ||\mathbf{x} - \mathbf{y}_i||^2 = \max_i{}^{-1} [\mathbf{x}^t \mathbf{y}_i + \alpha_i]$$

where $\alpha_i = -||\mathbf{y}_i||^2/2$ and the values of α_i can be precomputed and stored along with the codebook, we have the following algorithm.

Alternate Nearest Neighbor Encoding Rule

```
Step 1. Set $f = f_0$, $j = 1$, and $i = 1$.
Step 2. Compute $F_j = \mathbf{x}^t\mathbf{y}_j + \alpha_j$
Step 3. If $F_j > f$, set $F_j \rightarrow f$. Set $j \rightarrow i$.
Step 4. If $j < N$, set $j + 1 \rightarrow j$ and go to step 2.
Step 5. Stop. Result is index $i$.
```

The resulting value of i gives the encoder output $\mathcal{C}(\mathbf{x})$ and the final value of f determines the distortion between $\mathbf{x}$ and $\mathbf{y}_i$ according to $d(\mathbf{x},\mathbf{y}_i) = ||\mathbf{x}||^2 - 2f$. The initial value, f_0, must be smaller than any expected distortion value and is usually set to zero.

In real-time implementations of VQ encoders, the preferred choice of search algorithm depends on the particular processor architecture. The earliest reported hardware implementation of VQ [306] implemented the nearest neighbor search algorithm and used off-the-shelf components and the Z80A, an 8-bit microprocessor. Most commercial applications of VQ for speech compression algorithms today are implemented on single-chip programmable signal processors. For example, one engineer using a Texas Instruments TMS32010, a first generation signal processor chip, found that the basic nearest neighbor algorithm is most efficient. Another engineer used the AT&T DSP32 processor (which has floating point arithmetic) and found that the alternate nearest neighbor algorithm was more efficient. Generally, the code vectors are read from ROM into the processor's RAM and the input vector is also stored in RAM. Each component of each vector is stored in a separate memory location in RAM. With current generation processors, one component of the code vector and the corresponding component of the input vector can be simultaneously copied to the two input registers of a hardware multiplier, then the multiplication is performed and the product is added to the accumulator. This sequence of events provides the basic operation needed for the alternate nearest neighbor algorithm and can be performed in a single instruction cycle on most DSP chips. If an instruction cycle requires τ seconds, neglecting overhead operations, the encoding of an input vector for a codebook of N k-dimensional vectors takes place in $kN\tau$ seconds Advanced processors of 1991 vintage operate with an instruction cycle as fast as 35 ns so that for a 10-dimensional codebook of size 1024 vectors. one codebook search can be performed in 360 μs, for a coding rate of 2800 input vectors per second or 28,000 input samples per second.

As an alternative to programmable processors, VQ can be performed with application-specific integrated circuits (ASICs). Several experimental VLSI chips have been proposed or implemented that were specifically designed for codebook searching with the squared Euclidean distance algorithm [89], [92], [51], [99], [90]. In particular, a pipeline architecture chip based on the nearest neighbor algorithm was implemented in [89] and applied to a single-board speech coder [92]. A systolic chip set based on the alternate nearest neighbor algorithm was designed and implemented in [51], [90]. In [99], the alternate nearest neighbor algorithm was modified to implement the weighted squared error measure of equation 10.3.11. To our knowledge, there is no commercial use so far of custom or semi-custom VLSI implementations of VQ. This appears to be due to the rapid advance of programmable processor technology and the large variety of limited volume applications where custom chips are not cost-effective. Nevertheless, future high volume applications based on standardized speech coding algorithms are likely to lead to ASIC implementations of sophisticated algorithms that include VQ.

Encoding Complexity and Efficient Search Algorithms

The NN encoding algorithm for a Voronoi quantizer given earlier can be viewed as an *exhaustive search* algorithm where the computation of distortion is performed sequentially on every code vector in the codebook, keeping track of the "best so far" and continuing until every code vector has been tested. For a codebook of size N, this requires N distortion evaluations and each such evaluation involves k multiplications and $k-1$ additions for the squared error distortion. Other distortion measures may have much higher computational demands. Frequently the codebook size needed in applications is very large. Furthermore, the vector rate f_v is rather high in typical communications systems and the number of distortion calculations that must be performed per unit time, given by Nf_v, implies a very demanding computational complexity, typically involving many millions of arithmetic operations per second.

These considerations motivate serious study of more efficient algorithms that yield the nearest neighbor code vector without requiring an exhaustive search through the codebook. Several approaches have been taken to circumvent the complexity problem in VQ. Some solutions are based on designing a vector quantizer whose codebook has a special structure that allows faster encoding while paying the price that the quantizer is suboptimal for the given input statistics, vector dimension, and resolution. In

Chapter 12 some of these techniques will be presented.

At this point, we consider only the issue of finding fast search algorithms for an arbitrarily specified VQ without compromising performance. The idea here is that the codebook is assumed to be prespecified and designed to be optimal for a given performance objective. Does there exist an efficient nearest neighbor encoding algorithm that can achieve reduced complexity compared to the exhaustive search algorithm while guaranteeing that it will always find the nearest neighbor code vector? The answer is indeed yes for polytopal VQs as we shall see. The viewpoint that leads to efficient algorithms stems from the secondary and tertiary structure of polytopal quantizers as discussed in Section 10.2. Recall, in particular, that a decision that an input lies in a particular cell can be based on a number of binary hyperplane tests.

Consider the two-dimensional quantizer shown in Figure 10.10. In the figure the six code vectors are indicated with dots and the labeled "hyperplane" decision boundaries (line segments) are indicated with thick lines which separate neighbor regions. Thin lines indicate extensions of the hyperplane segments to help visualize the efficient encoding operation. Each

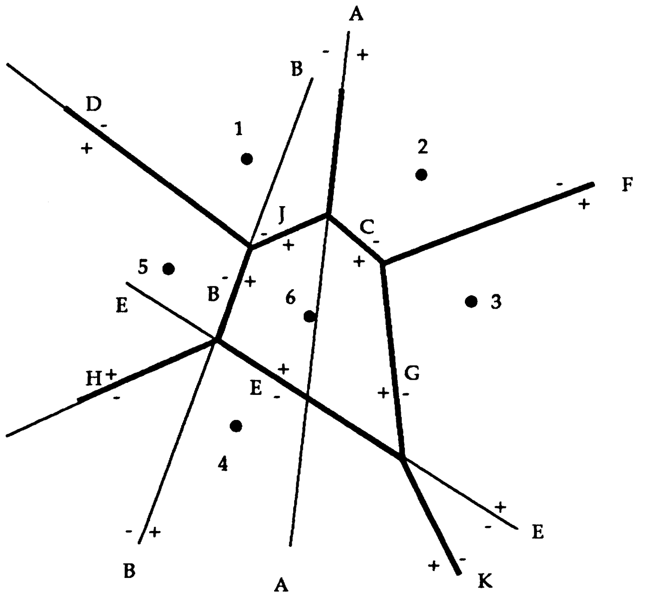

Figure 10.10: Two-Dimensional Quantizer

decision boundary is a segment of a hyperplane and is labeled with the letters $A, B, C, \cdots$ and the two half-spaces separated by each hyperplane are labeled ' + ' and '−'. We now describe a tree structure for an efficient successive approximation procedure to locate an input vector by a series of hyperplane decision tests. Suppose the initial step of a search algorithm is to compute the binary decision function for the hyperplane A. If the input $\mathbf{x}$ is on the right (labeled +) side of A, then code vector 1 and 5 are immediately eliminated as candidates for the nearest neighbor since they are contained entirely on the left side of A. Similarly, code vectors 2 and 3 are eliminated as candidates if the input is on the left side (−) of A. Depending on the result of the first test, we choose one of two new tests to perform. Thus if the input lies on the + side of A, we then test to see on which side of the hyperplane C it lies. If the input is on the − side of A, we then determine on which side of hyperplane B it lies. Each test eliminates one or more candidate code vectors from consideration. This kind of procedure corresponds to a tree structure for the search algorithm, as shown in Figure 10.11. In the figure each node is labeled with a letter corresponding to a particular binary decision test for the hyperplane (line) labeled in Figure 10.10. The tree is said to be *unbalanced* since different paths through the

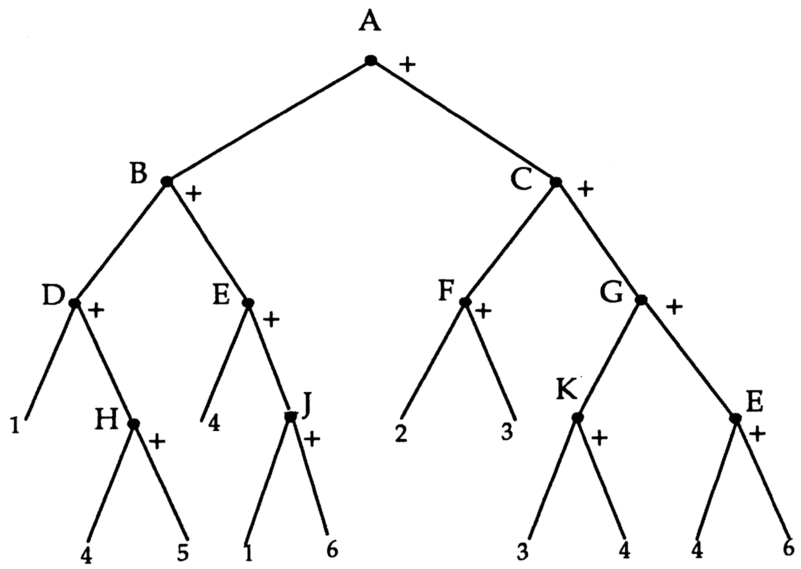

Figure 10.11: Tree Structure for efficient nearest neighbor search.

tree do not all have the same number of nodes. Each node of the tree where a hyperplane test occurs is labeled to identify the corresponding hyperplane in Fig. 10.10. The terminal nodes of the tree identify the final code vectors.

Note that this particular tree always leads to a final determination of the nearest neighbor code vector after at most 4 binary decisions. Hence, the complexity of the search has been reduced from 6 distance computations to 4 scalar product operations (neglecting the cost of performing comparisons). In less trivial cases where the codebook size is much larger than 6, it often turns out that a single hyperplane decision can eliminate a large subset of candidate code vectors. The maximum depth of the tree can be much less than the codebook size N. Furthermore, often the average complexity is much less than the worst case complexity of the tree search. In two dimensions, it is quite simple to see how to design the search tree from examination of the Voronoi cells for the given codebook. In higher dimensions, however, it is not so simple. In [65], a technique is described for finding an efficient tree structure for any given VQ codebook with any vector dimension. It was found that significant complexity savings can be achieved for searching an arbitrary codebook if the codebook size satisfies $N >> 2^k$, i.e., as long as the resolution r is somewhat larger than one bit per vector component. The reader should be warned, however, that complexity is not adequately measured by counting multiplies and ignoring comparison operations. Often simple logic operations can be more costly than multiply-adds.

Alternative methods exist for fast VQ encoding, however, they frequently require that the codebook itself be specially designed to allow a tree structure and hence, the performance achievable is not quite as good as with an optimal codebook. Alternative fast search methods will be described in Chapter 12. A systematic design technique for codebooks with balanced tree structures is given in Chapter 12 and for unbalanced tree structures is given in Chapter 17.

10.5 Lattice Vector Quantizers

Until this point we have retained considerable generality in discussing the nature and structure of VQ. A special class of vector quantizers that are of particular interest because of their highly regular structure are the so-called lattice quantizers. In one-dimensional quantization we have found that the codebook for a uniform quantizer is formed by truncating a lattice, where a lattice was defined as a set of equidistant points extending over the entire real line. Here we wish to consider multidimensional lattices. The concept of a lattice in R^k is identical to that presented in Chapter 3 on multidimensional sampling. The basic definitions are repeated here for the reader's convenience.

A lattice is a regular arrangement of points in k-space that includes the origin or zero vector, $\mathbf{0}$. The word "regular" means that each point "sees" the same geometrical environment as any other point. In other words, any translation of the set of points by subtracting one lattice point from all points in the lattice will yield the same lattice. A lattice Λ in $\mathcal{R}^k$ is a collection of all vectors of the form

$$\sum_{i=1}^{n} m_i \mathbf{u}_i, \tag{10.5.1}$$

where m_i are arbitrary integers and $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n$ are a linearly independent set of $n \leq k$ vectors. The most common case of interest is $n = k$, which we call *nondegenerate*. The set of vectors $\{\mathbf{u}_i\}$ is called a *basis* or *generating set* for the lattice Λ. Observe that if Λ is a lattice, then it has the property that if any two points $\mathbf{x}$ and $\mathbf{y}$ are in Λ, then $m\mathbf{x}+n\mathbf{y}$ is in Λ for any pair of integers (positive, negative, or zero) m and n. It can be shown (see Problem 10.12) that this is an equivalent definition of a lattice. It is convenient to define the *generating matrix* of a lattice, $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n]$. Since the vectors are linearly independent, $\mathbf{U}$ is nonsingular in the nondegenerate case where $n = k$. A well-known result of multidimensional geometry is that the volume of the parallelopiped formed by the basis vectors of a nondegenerate lattice is given by $|\det\mathbf{U}|$, the absolute value of the determinant of $\mathbf{U}$. If we view a lattice Λ as the codebook for a vector quantizer (with $N = \infty$), the Voronoi or NN cell of the lattice point 0 is given by $V = \{\mathbf{x} : ||\mathbf{x} - \mathbf{0}|| \leq ||\mathbf{x} - \mathbf{y}|| \text{ for each } \mathbf{y} \in \Lambda\}$. Since the NN cells form a partition of the space with the same density (cells per unit volume) as the partition formed by the parallelopiped cells, it follows that the volume of the NN cell is also equal to $|\det\mathbf{U}|$.

The reciprocal or dual lattice to a given nondegenerate lattice Λ with generating matrix $\mathbf{U}$ is that lattice whose generating function is given by $(\mathbf{U}^{-1})^t$, the transpose of the inverse matrix of $\mathbf{U}$.

Consider in particular the two-dimensional lattice Λ^2 whose generating matrix is given by

$$\mathbf{U} = \begin{bmatrix} 0 & \sqrt{3} \\ 2 & 1 \end{bmatrix} \tag{10.5.2}$$

so that each point $\mathbf{x} = (x_1, x_2)$ in Λ has coordinates of the form

$$x_1 = \sqrt{3}m_2$$

$$x_2 = 2m_1 + m_2$$

where m_1 and m_2 take on all integer values. Figure 10.12 shows the resulting arrangement of lattice points in the plane. Observe that the Voronoi cells are regular hexagons and the parallelopiped cell is a parallelogram.

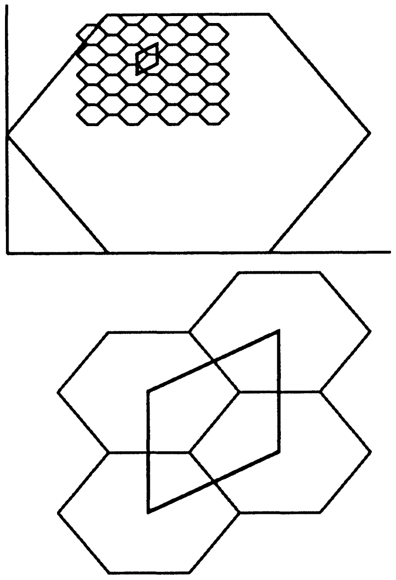

Figure 10.12: Two-Dimensional Hexagonal Lattice

In 3 dimensions there are several interesting and geometrically distinct lattice patterns which can be characterized by specifying the Voronoi cell shape. Many such patterns have been studied in crystallography. In particular, an important lattice in $\mathcal{R}^3$ has for its Voronoi cell the truncated octahedron which, as we shall see in Chapter 11, has certain optimal properties for quantization of uniform random vectors in 3 dimensions. The dodecahedron defines another three-dimensional lattice which occurs in the honeycombs of bees. A *lattice vector quantizer* is simply a vector quantizer whose codebook is either a lattice or a coset of a lattice or a truncated version of a lattice or its coset so that the codebook size is finite. (A coset of a lattice is the set of points formed by adding a fixed vector to each point in the lattice.) Clearly lattice quantizers are natural choices when uniform quantization is desired.

An interesting feature arises when the tertiary structure is applied to NN encoding of a lattice VQ. Let $\mathbf{n}_i$ denote the k-dimensional vector of integer components used to form the code vector $\mathbf{y}_i$,

$$\mathbf{y}_i = \mathbf{U}\mathbf{n}_i \tag{10.5.3}$$

for a lattice Λ. Then the $N \times k$ transform matrix $\mathbf{Y}$ in the tertiary structure can be written as

$$\mathbf{Y} = \mathbf{N}\mathbf{U}^t \tag{10.5.4}$$

where $\mathbf{N}$ is the $N \times k$ matrix whose ith row is $\mathbf{n}_i$. Thus the transformed variables to be scalar encoded in the tertiary structure are given by

$$\mathbf{a} = \mathbf{Y}\mathbf{x} = \mathbf{N}\mathbf{U}^t\mathbf{x}. \tag{10.5.5}$$

However, $\mathbf{N}$ is merely an integer matrix, so that the only scalar product operation that requires nontrivial arithmetic for computation is the formation of the k vector $\mathbf{U}^t\mathbf{x}$. This means that the main computational task for a NN lattice vector quantizer is to perform k scalar products of the input vector $\mathbf{x}$ with the basis vectors of the lattice. Once this is done, only integer linear combinations of the scalar product values must be formed and then scalar encoded with one-bit encoders and finally combinational logic produces the correct index. The great simplification in the encoding complexity that results from lattices is of course due to the regularity of the code vector locations and specifically due to the fact that each NN cell has the same set of faces apart from a translation of coordinates.

The preceding discussion is not meant to offer a specific encoding algorithm but rather merely to demonstrate the reduction in complexity due to the regularity of the code vector locations and also to show that the tertiary structure can be conceptually helpful. Efficient explicit algorithms for lattice encoding must depend on the specific lattice. See Problem 10.4 for a two-dimensional case of some interest. For a more extensive discussion of lattice encoding algorithms see [76] [75] [78] [79] [283] [148].

As a final note, we point out that the encoding algorithms for lattices are not directly applicable to vector quantizers based on truncated lattices. The difficulty arises in the boundary regions where the regularity of the lattice structure is no longer followed.

10.6 High Resolution Distortion Approximations

In this section we sketch the extension of the high resolution distortion approximations from scalar quantization to vector quantization. Here the results are more complicated and we only briefly describe the types of results that are available, confining interest to the squared error distortion. Additional details and developments for more general distortion measures may be found in [347] [348] [349] [137] [345] [159].

Given a vector quantizer with partition cells R_i and reproduction vectors $\mathbf{y}_i$, $i = 1, \cdots, N$, the average distortion can be expressed as in (10.3.3) by

$$D = \sum_{i=1}^{N} \int_{R_i} f_{\mathbf{X}}(\mathbf{x}) ||\mathbf{x} - \mathbf{y}_i||^2 \, d\mathbf{x},$$

where $||\cdot||$ denotes the Euclidean distance. As in the scalar high resolution development, we confine interest to the granular region and assume that the total contribution to the average distortion from the overload region is small. If N is large, the cells R_i small, and the probability density smooth, this permits us to make the approximation

$$D \approx \sum_{i=1}^{N} f_i \int_{R_i} ||\mathbf{x} - \mathbf{y}_i||^2 \, d\mathbf{x}$$

where we are assuming that $f_X(x) \approx f_i$ for $\mathbf{x} \in R_i$. We define

$$P_i = \Pr(X \in R_i) \approx f_i V(R_i),$$

where $V(R)$ is the volume of the set R (which is finite since we are in the granular region). Thus

$$D \approx \sum_{i=1}^{N} \frac{P_i}{V(R_i)} \int_{R_i} ||\mathbf{x} - \mathbf{y}_i||^2 \, d\mathbf{x}. \qquad (10.6.1)$$

Unlike the scalar high resolution results, (10.6.1) is not immediately useful because there is no easy way to approximate the integral for the arbitrary shapes possible for the cells R_i. The integral can be immediately recognized as the moment of inertia of the set R_i about the point $\mathbf{y}_i$. This at least provides a simple lower bound to the average distortion since the moment of inertia of an arbitrary set about an arbitrary point can be bounded below by the moment of inertia of a k-dimensional sphere of the same volume about its center of gravity. This approach leads to a family of bounds which can themselves be optimized over possible densities of quantizer levels. By generalizing the idea of a quantizer point density of Chapter 5 to vectors, applying multidimensional approximation arguments analogous to those used to obtain (5.6.9), and optimizing over the point density function it can be shown that in the high resolution regime,

$$D \geq \frac{k}{k+2} \left(\frac{2\pi^{\frac{k}{2}}}{k\Gamma(\frac{k}{2})} \right)^{-\frac{2}{k}} \left[\int f_X(\mathbf{x})^{\frac{k}{k+2}} \, d\mathbf{x} \right]^{\frac{k+2}{k}} N^{-\frac{2}{k}}, \qquad (10.6.2)$$

where $\Gamma(r)$ is the usual gamma function and k is the vector dimension.

This results will occasionally be quoted for comparison with the actual performance of some of the design examples to be presented in later chapters.

If the quantizer is a lattice quantizer and hence can be considered as a multidimensional uniform quantizer, then all of the cells are congruent.

If we further assume that the reproduction vectors are the centers of gravity (centroids) of the cells (the optimum choice), then all of the moments of inertia are the same. Assuming that the zero vector **0** is one of the reproduction vectors, (10.6.1) becomes

$$D \approx \frac{1}{V(R_0)} \int_{R_0} ||\mathbf{x}||^2 \, d\mathbf{x} = \frac{1}{|\det \mathbf{U}|} \int_{R_0} ||\mathbf{x}||^2 \, d\mathbf{x}, \qquad (10.6.3)$$

where R_0 is the nearest neighbor or Voronoi cell about the point **0** and **U** is the generator matrix for the lattice. This quantity has been tabulated for many popular lattices [78] [76] [75] [79] and hence the high resolution approximation does give simple expressions for the average distortion for lattices.

10.7 Problems

10.1. A three point (N=3) quantizer in two dimensions has the codebook given below. If the input vectors are two-dimensional random variables uniformly distributed on the unit square: $0 < X < 1$, $0 < Y < 1$, determine the mean squared quantizing error by Monte-Carlo generating 150 independent random vectors with this distribution and averaging the squared error obtained in each case. You should turn in a printout of your source code and data output.

Point	x-comp	y-comp
1	0.25	0.25
2	0.75	0.50
3	0.40	0.75

Table 10.1: Codebook

10.2. A pattern classifier operates on 3-dimensional input vectors $\mathbf{x} = (x_1, x_2, x_3)$ and performs the following classification rule: If $x_1 > 0$ and $x_2^2 + x_3^2 < 1$, then the pattern is in class 1; if $x_1 < 0$, then the pattern is in class 2; if neither of these conditions is satisfied, the pattern is in class 3. Explain why this classifier is a valid example of a VQ encoder. Is it regular? Explain.

10.3. A VQ encoder is defined by a table consisting of N pairs of vectors $(\mathbf{y}_i, \mathbf{z}_i)$ for $i = 1, 2, \ldots, N$. The encoder computes the distortion values $d_i = ||\mathbf{x} - \mathbf{y}_i||^2$ for $i = 1, 2, \ldots, N$ and $d_{N+j} = ||\mathbf{x} - \mathbf{z}_j||^2$

for $j = 1, 2, \ldots, N$. Then the encoding rule assigns index m, for $m = 1, 2, \ldots, N$, to the given input vector $\mathbf{x}$ when either $d_m \leq d_i$ for $i = 1, 2, \ldots, 2N$ or $d_{N+m} \leq d_i$ for $i = 1, 2, \ldots, 2N$ and a suitable tie-breaking rule is used.

(a) Sketch a typical shape of the partition for a simple 2-dimensional example.

(b) Is this VQ regular? Explain.

10.4. (a) Derive an expression for the computational complexity needed to encode a k-dimensional random variable with a b bit codebook (containing 2^b k-dimensional code vectors), each component of each vector is a real number stored to a suitably high accuracy. The result of an encoding is simply the index identifying which vector in the codebook is the best representation of the input vector. Assume the cost of additions or comparisons is negligible but a scalar multiplication of two real numbers is the unit for measuring complexity.

(b) Repeat part (a) but now suppose that an addition costs 3 units, a scalar multiplication costs 23 units, and a comparison between two real numbers costs 1 unit.

In both parts you must explain how you derive your result.

(c) If one unit in part (b) requires 0.1 microseconds of a processor's time, and only one processor is used to implement the encoder, how many vectors can be encoded per second when $k = 8$ and $b = 16$?

10.5. A digital signal processor (DSP) with a multiply-add cycle time of 100 ns is used to quantize speech samples arriving at 8 kHz rate. The DSP sends out an encoder bitstream at 8 kb/s. Ignoring whatever overhead might be necessary to implement an encoder and considering just VQ complexity, give an estimate of the maximum block size that you can use for encoding consecutive blocks of k contiguous samples by VQ with an arbitrary unstructured codebook. Explain.

10.6. A two-dimensional random vector $\mathbf{X}$ is uniformly distributed in the X_1, X_2 plane over the square centered at the origin, with sides parallel to the coordinate axes, and of length 10. A nearest neighbor vector quantizer $Q_{\mathbf{X}}$ for $\mathbf{X}$ has codebook given by

$$\mathcal{C}_{VQ} = \{(2,5), (-2,3), (1,4), (3,4), (-3,-1), (2,0)\}$$

Alternatively a pair of nearest neighbor scalar quantizers Q_1 and Q_2 for separately quantizing X_1 and X_2 are given by the codebooks $C_1 = \{-2, +2\}$ and $C_2 = \{-3, 0, +3\}$.

(a) If the input to the vector quantizer is the vector $(-1, 4)$ find the quantized output vector and the mean squared error.

(b) If x_1 is quantized with the scalar quantizer whose output levels are (-2, +2) and x_2 is quantized with a scalar quantizer whose output levels are (-3, 0, +3), where $\mathbf{x} = (x_1, x_2)$ is as given in part (a), find the reconstructed output of the quantizer when the input is as given in part (a). Explain.

(c) Find the equivalent VQ codebook for the scalar quantization case of part (b). In other words, find the set of all possible reconstructed vectors that can be reproduced by scalar quantizing the components according to (b).

(d) In general (without reference to the specific numbers given in this problem), compare qualitatively the performance achievable for the two quantization methods for the input random vector. Assume the same total number of bits for both methods.

10.7. Each code vector of a two-dimensional quantizer is defined by $x = m\sqrt{3}$ and $y = m+2n$ where m and n take on all possible integer values so that the number of output points is infinite. (This approximation is made to neglect boundary effects.)

(a) Sketch the Voronoi cells in the plane for a few points near the origin.

(b) Find a simple algorithm that will identify the nearest code vector to any given input point in the plane.

(c) Find the nearest lattice point to $x = (374.23, -5384.71)$.

10.8. The following technique leads to a table that has been proposed for fast search through a VQ codebook. Let x_1 denote the first component of the k dimensional vector $\mathbf{x}$ in the k dimensional space $\mathcal{R}^k$. For a given vector quantizer with partition regions R_k, find the smallest value s_i and biggest value b_i of the x_1 component of all points lying in region R_i. Then sort all the values of s_i and b_i into a sequence of increasing numbers: $u_1 \leq u_2 \leq u_3 \leq \ldots \leq u_m$ and for each pair of adjacent values u_j, u_{j+1} record the indices of all regions of the partition which contain point with x_1 component lying in this interval. Once this data is computed and stored, a fast search algorithm can be achieved. Describe a fast search algorithm that will make use of these tables to find the nearest neighbor region for a given input vector without performing an exhaustive search through the codebook. Illustrate the main idea of this method by sketching a VQ partition in 2 dimensions and indicating the role of the values u_i in the figure. Explain.

10.9. Consider a k-dimensional nearest neighbor vector quantizer with a given codebook of size N. Prove the following: The encoder can be implemented by augmenting the input vector $\mathbf{x}$ (dimension k) by adding a $(k+1)$th component with value 1 and then simply finding the maximum scalar product between the augmented input vector and each vector in a modified codebook where each vector has dimension $k+1$ and the codebook size N remains the same. Specify how the new codebook is generated from the old codebook.

10.10. Define the taxicab distance measure as

$$d(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{k} |x_i - y_i|$$

(a) Sketch the region in the plane for the 2-dimensional VQ case consisting of all points having a taxicab distance measure less than 2 from the origin.

(b) Sketch the boundary separating the set A of all points in the plane that are closer (in the sense of the taxicab distance measure) to (-1,0) than to (3,0) from the set B of all points closer to (3,0) than to (-1,0).

(c) In general, for a given VQ codebook, how would you compare the partition regions for a nearest neighbor partition for the case of squared error distance and taxicab distance?

10.11. For VQ based on the Euclidean distance measure, let two codevectors $\mathbf{y}_1$ and $\mathbf{y}_2$ be separated by a distance H and suppose an input vector $\mathbf{x}$ is distance A from $\mathbf{y}_1$ and distance B from $\mathbf{y}_2$. Show that if $H > 2A$, then $A < B$, so that it is not necessary to compute the value of B in order to do a nearest neighbor search for the input $\mathbf{x}$.

10.12. Prove the statement made in Section 10.5 that a collection of vectors $\Lambda = \{\mathbf{x}_i\}$ is a lattice if and only if for any $i \neq j$, the points $m\mathbf{x}_i + k\mathbf{x}_j \in \Lambda$ for all integers m and n.

10.13. Evaluate the high resolution approximation (10.6.3) for the lattice of (10.5.2) (see also Problem 10.7). Repeat for the dual lattice.