**EXAMPLE OF SCALAR AND VECTOR QUANTIZATION**

**Source sequence** $x_k$:
This could be the output of a highly correlated source.

**A scalar quantizer:**

$k = 1, \ M = 4$

$C_1 = \{w_1, w_2, w_3, w_4\} = \{-4, -1, 1, 4\}$

= codebook of quantization levels

levels and cells shown

$R = 2 \text{ bits/samp}$

The scalar quantizer applied to the source sequence:

The open circles show the quantized outputs

**A Vector Quantizer:**

$k = 2, \ M = 10, \quad R = \frac{1}{2} \log_2 10 = 1.66 \text{ bits/sample}$

$C_2 = \{(-4,-4), (-4,-1), (-1,-4), (-1,-1), (-1,1), (1,-1), (1,1), (1,4), (4,1), (4,4)\}$

The codevectors and quantization cells

The idea behind this VQ:
When the scalar quantizer is applied to a slowly changing data sequence, successive source samples are usually quantized into identical or adjacent "levels". The codebook $C_2$ consists of just those pairs having identical or adjacent components. In comparison, the scalar quantizer permits any of the 16 possible pairs of successive levels in the set

$C_1 \times C_1 = \{(-4,-4), (-4,-1), (-4,1), (-4,4), (-1,-4), (-1,4), (-1,1), (-1,4), (1,-4), (1,4), (1,1), (1,4), (4,1), (4,4)\}$.

Since $10 < 16$, the VQ has lower rate than the scalar quantizer.

The result of the VQ applied to the source sequence is shown

Lines connecting open circles indicate chosen codevectors.
The distortion and rate for the scalar and vector quantizers:

\[ R = \frac{1}{2} \log_2 4 = 1 \text{ bit/sample} \]

(The distortions shown are not actual MSE, just representative values.)

Another vector quantizer:

\[ k = 2 \]
\[ M = 4 \]
\[ C_3 = \{ (-4,-4), (-1,-1), (1,1), (4,4) \} \]

The codevectors and quantization cells

This VQ applied to the source sequence is shown

Lines connecting open circles indicate chosen codevectors.
OVERLOAD AND GRANULAR REGIONS AND DISTORTION

For a typical quantizer, one may subdivide k-dimensional space roughly into two regions:

- The overload region \( R_o \), consisting of all \( x \) such that \( \|x - Q(x)\|^2 \gg D \)
- The granular region \( R_G \), consisting of all \( x \) such that \( \|x - Q(x)\|^2 \lesssim D \)

There is no widely accepted precise definition of overload and granular regions, but roughly speaking the overload region is where distortion is large, i.e. where the quantizer is said to be "overloaded", and the granular region is where the quantization errors are small, i.e. the "noise" added by quantization is "granular".

It is often useful to decompose the distortion into granular and overload distortions. That is,

\[ D = D_G + D_O \]

where

\[ D_G = \text{granular distortion} = \frac{1}{k} \int_{R_G} \|x - Q(x)\|^2 f_X(x) \, dx \]

\[ D_O = \text{overload distortion} = \frac{1}{k} \int_{R_O} \|x - Q(x)\|^2 f_X(x) \, dx \]

Typically, when a quantizer is designed to have small distortion, \( D_o \ll D_G \)

KEY QUESTIONS:

- How to implement the encoder, i.e. the partitioning?
- Complexity?
- How to design/optimize fixed-rate VQ's?
  (What properties do good fixed-rate VQ's have?)
- How to estimate MSE of a VQ?
- How to design low complexity VQ's with good performance?
- What is best possible performance (D vs. R) of a VQ? (the opta function) How does it depend on dimension k?
- How well do low complexity VQ's perform? What is it in their structure that limits their performance?

OUTLINE OF VQ COVERAGE

- Optimality properties of fixed-rate VQ's.
- "Full search" encoding.
- Generalized-Lloyd iterative VQ design algorithms
- Properties of optimal quantizers, e.g. \( E \|Y\|^2 = E \|X\|^2 - E \|X-Y\|^2 \)
- High-Resolution Analysis of MSE -- Bennett's integral for VQ
- High-resolution analysis of optimal performance -- Zador-Gersho formula
- Comparison to Shannon's rate-distortion theory analysis of optimal performance
- Variable-rate VQ -- optimality properties and high-resolution theory.
THE LBG ALGORITHM

We wish to design a k-dimensional VQ with size $M$.

Training Sequence: This algorithm presumes we have a representative sequence of k-dimensional vectors from the source, $t_1, t_2, \ldots, t_N$, called a training sequence of training vectors. If the source density is known, the training sequence could be generated by a random number generator. More commonly, one takes a long sample from the source to be quantized.

**LBG Algorithm:**

- Choose an initial codebook $C^{(0)} = \{w_1^{(0)}, \ldots, w_M^{(0)}\}$.
- Iterate the following steps until the centroids change little and/or the distortion changes little.

Given $C^{(i)} = \{w_1^{(i)}, \ldots, w_M^{(i)}\}$ produced by the $i$th iteration,

1. Partition training set: $S^{(i+1)} = \{S_1^{(i+1)}, \ldots, S_M^{(i+1)}\}$

   $$S_j^{(i+1)} = \{ t_n : t_n \text{ closer to } \tilde{w}_j^{(i)} \text{ than to } \tilde{w}_{j'}^{(i)}, j' \neq j \}$$

2. Find empirical centroids:

   $$\tilde{w}_j^{(i+1)} = \frac{1}{N_j^{(i+1)}} \sum_{n : t_n \in S_j^{(i+1)}} t_n,$$

   where $N_j^{(i+1)} = \# \text{ training vectors in } S_j^{(i+1)}$.

In Step 1, for each training vector $t_n$, we apply full search encoding to find the closest codevector in $C^{(i)}$. The training vector and the index of the closest codevector are stored in a table (an $N \times 2$ array).

Step 2 is implemented by counting and averaging all training vectors with a given label.

Alternatively, one may combine these two steps into the following single step:

3. Given $C^{(i)} = \{w_1^{(i)}, \ldots, w_M^{(i)}\}$ produced by the $i$th iteration,

   $$\tilde{w}_j^{(i+1)} = \frac{1}{N_j^{(i)}} \sum_{n : t_n \text{ closest to } \tilde{w}_j^{(i)}} t_n,$$

   where $N_j^{(i)} = \# \text{ training vectors closest to } \tilde{w}_j^{(i)}$.

In other words, for each training vector $t_n$, one finds the closest codevector, say $\tilde{w}_j^{(i)}$, to which it is closest, via full search. One increments a counter that stores $N_j^{(i)}$, and one adds $t_n$ to an accumulator that computes the sum for $\tilde{w}_j^{(i)}$. 

Issues:
1. Initial codebook: Same options as before.
2. Basic iteration: Already discussed.
3. Stopping conditions: Same options as before.
4. Convergence:
   As before, each step maintains or decreases the \textit{training distortion}:
   \[
   D_{\text{train}} = \frac{1}{N} \sum_{n=1}^{N} ||t_n - Q(t_n)||^2
   \]
   Therefore, the training distortion converges to some value. Usually, the quantizer converges to a local optimum, i.e. to a quantizer that satisfies the empirical versions of (*) and (**).
   Moreover, since there are only a finite number of distinct partitions of the given training sequence, after a finite number of steps the algorithm necessarily reaches a local optimum after which it does not change, or it reaches a limit cycle, i.e. it cycles repeatedly through some finite number of partition-codebook pairs. However, the algorithm is usually stopped long before either of these occur.
   Because the algorithm works with a finite set of training vectors, there tend to be more local optima than with algorithms that work directly with the pdf such as the first generalized Lloyd algorithm. Therefore, it is usually wise to rerun the algorithm with several different choices of initial codebook.

5. Software
   We have a version of the LBG algorithm available in C. Other versions can be found on the web.

6. Alternate derivation of the LBG algorithm:
   Instead of hypothesizing an unknown pdf, suppose that we know only that we have a training sequence \(t_1, t_2, \ldots, t_N\), and that we desire to choose codevectors \(w_1, w_2, \ldots, w_M\) to minimize the training distortion
   \[
   D_{\text{train}} = \sum_{n=1}^{N} \min_{j} ||t_n - w_j||^2
   \]
   To attempt to do this, repeatedly perform the following:
   \[
   \tilde{w}_j^{(i+1)} = \frac{1}{N_j^{(i)}} \sum_{n : t_n \text{ closest to } \tilde{w}_j^{(i)}} t_n
   \]
   where \(N_j^{(i)} = \# \text{ training vectors closest to } \tilde{w}_j^{(i)}\)
   Performing the above will not increase and usually decrease
   \[
   \sum_{n=1}^{N} \min_{j} ||t_n - w_j||^2
   \]
7. Training distortion vs. actual distortion:

Suppose that the LBG algorithm is run on a training sequence \( t_1, t_2, \ldots, t_N \) taken from a source with probability density \( p(x) \), and it designs a quantizer \( Q \). Then, the actual distortion

\[
D = E((X-Q(X))^2)
\]

of \( Q \) will usually be greater than the training distortion,

\[
D_{\text{train}} = \frac{1}{N} \sum_{n=1}^{N} ||t_n - Q(t_n)||^2
\]

which is the distortion measured on the training sequence.

To see that actual and training distortion can be different, consider the extreme case where the training sequence length \( N \) equals the size \( M \) of the desired VQ. In this case, the algorithm will choose the codebook to be the training sequence itself, and it will find the training distortion to be zero, whereas the actual distortion will definitely be larger. Moreover, the designed quantizer is probably far from optimal.

As illustrated on the next page, the difference between training and actual distortions usually becomes smaller as \( N \) increases.

Because training distortion can be substantially less than actual distortion, it is customary to estimate the actual distortion by running the VQ on a test sequence that is distinct from the training sequence.

Training sequence length:

How large to make the training sequence length \( N \)?

Note: As illustrated below, the VQ designed by the algorithm usually becomes better as \( N \) is made larger, and the training distortion becomes a more accurate estimate of actual distortion.

A typical rule of thumb is that \( N \) should be at least \( 50M \), and larger is better. Larger \( N \) is needed if training distortion is to be used as an estimate of actual distortion. (Actually, the size of \( N \) should also increase with dimension \( k \).)

Another strategy, which is fairly conservative, is to choose \( N \) large enough that the training and test sequence distortions are reasonably close. In this case, we can be pretty sure we have both a good quantizer and a good estimate of actual distortion.