Quick Introduction to
A Variety of Lossy Source Coding Techniques

Generic Quantization Techniques
- Fast VQ partitioning
- Structured VQ
  - Tree-structured VQ (TSVQ)
  - Two-stage and multi-stage quantization
- Lattice VQ
- Shape-gain, Polar and Pyramid quantization
- Transform, subband and wavelet coding
- Hierarchical table lookup VQ

Nonblock coders
- Differential pulse code modulation (DPCM)
- Trellis quantization

Techniques for
- Image coding
- Video coding
- Speech coding
- Audio coding

Fast Partitioning for Unstructured VQ
AKA Fast Search of Unstructured VQ Codebooks

Here we describe several fast methods for finding the closest codevector in a
k-dimensional codebook \( C = \{ w_1, \ldots, w_M \} \) to a given vector \( x \). All but the first of
these obtain reduced computational complexity at the expense of increased
storage.


1. Partial Distortion

Suppose we have found that the closest codeword to \( x \) among \( w_1, \ldots, w_{n-1} \) is
at distance \( d \) from \( x \), and suppose we are now computing

\[ ||x-w_n||^2 = (x_1-w_{n,1})^2 + (x_2-w_{n,2})^2 + (x_3-w_{n,3})^2 + \ldots \]

by successively computing and accumulating the square terms.

If after some number of terms have been accumulated this sum should
become larger than \( d^2 \), then we know \( w_n \) is not the closest codeword, so we
need not compute any more terms, but move on to computing \( ||x-w_{n+1}||^2 \).

Since most codewords are quite far from \( x \), this method frequently reduces
the number of op's/sample by 25 to 75%.
2. Successive Narrowing of Search

Initialization: The distances between all pairs of codevectors are precomputed and stored in a table.

a. Choose an initial codevector \( y \).

b. Eliminate all codevectors \( w_i \) such that
\[ ||w_i - y|| > 2 ||x - y||. \]

By the triangle inequality
\[ ||w_i - y|| \leq ||w_i - x|| + ||x - y||, \]
so that
\[ ||w_i - x|| \geq ||w_i - y|| - ||x - y|| > ||x - y||. \]
\( \Rightarrow w_i \) could not be closest codeword.

c. Successively search the codevectors not eliminated to find one closer to \( x \) than \( y \), which then replaces \( y \).

d. Eliminate all remaining codevectors such that \( ||w_i - y|| > 2 ||x - y|| \).

e. Repeat steps c. and d. until all codevectors have been considered or eliminated from consideration.

This method reduces # ops/sample at the expense of storage.

Question: How to implement Step b efficiently?

There are many variations of and improvements to this method.
As far as I know, the complexity still increases exponentially with \( k \) and \( R \).

3. Coarse-to-Fine Search

In this method, a very fast coarse quantization is done first. Then a full search is performed on a small subset of \( C \).

Initialization: Select a scalar quantizer. Make a table with one row for each cell of the k-fold product quantizer. (Such cells are rectangles.) The row corresponding to a given cell of the k-fold product quantizer contains a list of the indices of all codevectors in \( C \) whose Voronoi regions intersect the given cell.

Operation:

a. Scalar quantize each component of \( x \).

b. Use the sequence of k scalar quantizer outputs to address the table and obtain a set of codevector indices.

c. Compute the distance between \( x \) and the codevectors whose indices were found in Step b.

d. Output the index of the cell whose codevector is closest.

There are variations on this method where Step a. is replaced by some other coarse quantization method, e.g. one of the fast structured VQ's to be discussed later.
4. Fine-to-Coarse Search

This is the logical continuation of method 3. This is like coarse-to-fine, except that the rate of the initial scalar quantizer is sufficiently higher than that of the desired VQ that most of the cells of the product quantizer are contained in one and only one Voronoi region of C. In this case, Step c. is eliminated in most cases, or reduced to just a few computations.

5. Hierarchical table lookup.
To be described later.

---

**STRUCTURED VECTOR QUANTIZATION**

These are VQ techniques that impose "structure" on the codebook that permits the partitioning to be done in a simpler fashion than brute force full search.

The codebooks of these techniques are suboptimal. For example, they may fail to satisfy one or both of the optimality criteria.

For many of these methods, the partition is not the Voronoi partition for the given codebook.

However, the "structure" often enables such a large reduction in complexity that VQ's with higher dimension become practical. These higher dimensional structured VQ's work better than optimal VQ's of the same complexity, which have smaller dimensions.

Here we just briefly introduce the techniques and how well they work. Later we'll analyze some of them, e.g. with Bennett's integral.
Tree-structured VQ (TSVQ) is a quantization method in which the quantizing partition is structured in a way that permits one to find the cell containing a source vector \( \mathbf{x} \) with few arithmetic operations.

Specifically, a fixed-rate, tree-structured VQ with dimension \( k \) and rate \( R \) is characterized by a complete binary tree with depth \( kR \). There is a "hyperplane" associated with each "internal node", and with a quantization cell associated with each "leaf". A complete binary tree of depth five is shown to the right. The leftmost node is called the "root" node. The rightmost nodes are called "leaves". All nodes except leaves are called "internal nodes". Each internal node has two "branches" leading from it to the right. The nodes to which these branches are connected are called the "upper child" and the "lower child" of the node.

A hyperplane is a set of the form \( H(\mathbf{v}, T) = \{ \mathbf{x} : \mathbf{x} \cdot \mathbf{v} = T \} \), where \( \mathbf{v} \) is the "normal" to the hyperplane, \( T \) is the displacement, and \( \mathbf{x} \cdot \mathbf{v} = \sum_{i=1}^{k} x_i v_i \). Without loss of generality, we assume \( \mathbf{v} \) is normalized to have unit length. If \( \mathbf{x} \cdot \mathbf{v} > T \), then \( \mathbf{x} \) is said to be "above \( H(\mathbf{v}, T) \)" and if \( \mathbf{x} \cdot \mathbf{v} < T \), then \( \mathbf{x} \) is said to be "below \( H(\mathbf{v}, T) \)"

A TSVQ has a hyperplane associated with each internal node of the tree. The leaves are number 1 to \( 2^{kR} \) from top to bottom. There is a codevector associated with each leaf.

The partitioning proceeds as follows. Let \( \mathbf{x} \) be the vector to be quantized. Let the root node be the "current" node. Let \( H_0 \) be the hyperplane of the root node. If \( \mathbf{x} \) lies above \( H_0 \), we "move" to the upper child of the root node; otherwise we move to the lower child. Let \( H_1 \) denote the hyperplane of the node to which we moved, which becomes the current node. If \( \mathbf{x} \) lies above \( H_1 \) move to the upper child of the current node; otherwise move to the lower child. In this manner, in \( kR \) moves we find ourselves at a leaf. The integer labelling this leaf is considered the index of the cell in which \( \mathbf{x} \) lies.

The cell is the intersection of the half spaces determined by the \( kR \) hyperplane tests.

The partition and codevectors of a typical TSVQ with \( k = 2, R = 2.5 \), and \( M = 2^{kR} = 32 \) is shown to the right.

Notes:
- Any scalar quantizer can be implemented as a tree-structured quantizer.
- A TSVQ partition is not a Voronoi partition, i.e. it is not optimal for its codebook.
**COMPLEXITY**

Encoding

Arithmetic complexity

Encoding requires $kR$ hyperplane tests to encode $x$.
Each hyperplane test requires $k$ multiplications, $k-1$ additions, and one comparison.
In total, encoding requires, approximately, $2k^2R$ operations.
That is, arithmetic complexity for encoding is

$$2kR \text{ ops/sample}$$
as compared to $3M = 3 \times 2kR$ ops/sample for full search encoding.

Storage

The encoder must store the hyperplane at each internal node.
There are $2^{kR-1}$ internal nodes in a binary tree with $2^{kR}$ leaves.
Storing a hyperplane requires storing $k+1$ numbers ($v$ and $T$).
Thus the storage required is approximately

$$k2^{kR}$$

numbers.
This is the same as with brute force full search encoding.

Summary

TSVQ requires the same storage, but much less computation than full search VQ.

**GREEDY DESIGN ALGORITHM**

1. Use the LBG algorithm to optimize a two-cell quantizer and let the hyperplane at the root node be the boundary between the two cells.
2. Each of these two cells is then associated with a child of the root node. For each of these children, use the LBG algorithm to optimize a two-cell quantizer for the training data that lies in the cell associated with that child. Let the hyperplane associated with the child be the boundary between the two cells found by the LBG algorithm.
3. Each of these cells is then associated with a child of the current node. For each child, use the LBG algorithm to optimize a two-cell quantizer for the training data that lies in the cell associated with that child. Let the hyperplane of the child be the boundary between the two cells found by the LBG algorithm.
4. Repeat Step 3 until a hyperplane has been found for each internal node of the tree.
5. Let the codevectors be the centroids of their respective cells.

Notes:

1. This algorithm does not produce an optimal TSVQ. Instead it makes greedy choices at each step.
2. Even if one found the best possible tree, the codebook would not be an optimal codebook, and the partition would not be optimal for the given codebook. However, the codebook will be optimal for the partition.
The following slides illustrate the design algorithm as it operates on a training set from a Gauss-Markov source with $\rho = .9$. 
GAUSS-MARKOV SOURCE, CORR. COEFF. $\rho = .9$

$k = 2$, $N = 128$, $R = 3.5$

$k = 2$, $N = 256$, $R = 4$
**Example of Predicted SNR for TSVQ**

- Gauss AR Source, $\rho = 0.9$
- $R = 3$

**Performance of TSVQ & Optimum VQ**

![Graph showing SNR vs. Rate for TSVQ and VQ*](image)

**Speech Samples**

- $k = 8$
- $k = 4$
- $k = 2$

**SNR**

- $\text{SNR}^*_k(R)$
- $\text{SNR}_{\text{TSVQ}}(k, R)$
- $\text{SNR}^*(k, R)$
- $\text{SNR}_{\text{TSVQ}}^*(k, R)$
OTHER TREE-STRUCTURED VQ'S

- M-ary trees
  + Complexity increases from $\sim 2kR$ to $\sim 3 \frac{M-1}{\log_2 M} kR$.
  + Point density would, apparently, not be improved.
  + Cell shapes might be improved.

- Variable-depth trees
  + Tree growing
  + Tree pruning

TWO-_STAGE VECTOR QUANTIZATION (2VQ)

- Operation:
  1. Quantize $X$ using $Q_1$ with rate $R_1$
  2. Quantize the error/residual $U = X - Y_1$ using $Q_2$ with rate $R_2$

- Rate: $R = R_1 + R_2$

- Distortion: $D = \frac{1}{K} \mathbb{E} ||U - V||^2 = $ Stage 2 distortion, because $(X - Y) = (U - V)$

- Complexity: much lower than single-stage VQ -- $M_1 + M_2$ vs. $M_1 M_2$

- Greedy Design:
  1. design $Q_1$ to minimize $\mathbb{E}||X - Y_1||^2$
  2. design $Q_2$ to minimize $\mathbb{E}||U - V||^2$, e.g. on training sequence of first-stage errors.
Performance: Example -- Gaussian AR source, \(\rho = .9\), VQ dimension \(k = 2\).

<table>
<thead>
<tr>
<th>Rate (R)</th>
<th>SNR (db)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30.48</td>
</tr>
<tr>
<td>2</td>
<td>20.00</td>
</tr>
<tr>
<td>3</td>
<td>10.00</td>
</tr>
</tbody>
</table>

Predicted and actual SNR for a Gaussian AR source, \(\rho = .9\), \(k = 2\).
EXAMPLE OF PREDICTED SNR FOR 2VQ

- Gaussian AR Source -- corr. coeff. $\rho = 0.9$
- $S_{\text{sq}}(k,R) = S(k,R) - 10 \log_{10} L, \ R = 3$

<table>
<thead>
<tr>
<th>$k$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L\text{dB}$</td>
<td>6.9</td>
<td>3.0</td>
<td>2.53</td>
<td>1.81</td>
</tr>
</tbody>
</table>

**SNR2vq,k(3)**

**SNR*(3)**

**SNRk(R)**

Calculated losses, the rest are fiction

VARIATIONS OF TWO-STAGE VQ

- Can use other types of lossy coders in two stages. (e.g. VQ & scalar Q, transform Q and DPCM)
- Multistage quantization: Two or more stages.
- Cell-Conditioned Two-Stage VQ
CELL-CONDITIONED TWO-STAGE VQ.

With a little extra complexity in the second stage, can make 2VQ work almost as well as single-stage VQ.

\[
\begin{array}{c}
\text{Stage 1} \quad Q_1, R_1 \quad Y_1 \\
\text{linear transformation} \\
\text{Stage 2} \quad U \\
\text{T}_1 \quad Z \\
\text{Q}_2, R_2 \quad \hat{Z} \\
\text{inverse transformation} \quad \text{T}_1^{-1} \\
\text{U} \\
\text{Y}_2
\end{array}
\]

It usually suffices for the transform to be a simple scaling and for the second stage quantizer to have uniform point density.

CC2VQ vs. 2VQ

Gaussian AR Source, $\rho = .9$

\[
\begin{align*}
k &= 2 \\
\text{SNR (dB)} & \quad \text{Rate R}
\end{align*}
\]

\[
\begin{align*}
k &= 4 \\
\text{SNR (dB)} & \quad \text{Rate R}
\end{align*}
\]
LATTICE (VECTOR) QUANTIZER

- A lattice is an infinite set of points in $\mathbb{R}^k$ that is closed under vector addition and subtraction.

  Equivalently, a lattice is generated by a basis $\{u_1, ..., u_k\}$ (set of linearly independent vectors). The lattice is
  \[ \{ w = \sum_{i=1}^{k} c_i u_i : c_1, ..., c_k \text{ are integers} \} \]

- A "finite" lattice VQ is a "contiguous" subset of the lattice.

- There are low complexity "algebraic" partitioning algorithms for many lattices (linear in $kR$)

- One also needs a low complexity indexing algorithm. In some special cases, there exists such.

- Lattice VQ's can also be used with variable-rate coding. But to my knowledge this has not been well explored. That is, we need low complexity methods of assigning binary codewords with good lengths to cells.

POLAR QUANTIZATION

- Independently quantize magnitude & angle

- Especially good for circularly symmetric source densities such as IID Gauss.

- As discussed earlier (Bennett, pp. 30-35), when optimized for IID Gaussian, this method has SNR $0.41$ dB larger than optimal scalar quantization, and $0.89$ dB less than optimal 2-dim'l quantization.

UNRESTRICTED POLAR QUANTIZATION

- Permit the number of quantized angles to vary with the quantized magnitude.

- When optimized for an IID Gaussian source, this method has SNR $1.13$ dB better than optimal scalar quantization, and only $0.17$ dB less than optimal 2-dim'l quantization.
Gain-Shape VQ

A generalization of polar quantization to dimension $k > 2$.

Component codebooks:
- $C_A = \{w_1, \ldots, w_{MA}\}$ = codebook of scalars for the "magnitude" also called the "gain"
- $C_S = \{v_1, \ldots, v_{MS}\}$ = shape codebook of $k$-dimensional vectors each with $||v_i|| = 1$

The gain-shape codebook:
- $C = \{w_i v_j : i = 1, \ldots, MA, j = 1, \ldots, MS\}$

Codebook size:
- $M = MA \times MS$

Code rate:
- $R = \frac{1}{k} \log MA \times MS = \frac{1}{k} (RA + RS)$

Encoding complexity:
- There is a fairly simple optimal partitioning algorithm, i.e. one that uses the Voronoi partition. Can you figure it out?

Pyramid VQ

- A variation of $k$-dimensional Gain-Shape VQ in which "magnitude" is measured with $\sum_{i=2}^{k} |x_i|$ instead of the usual $\sum_{i=1}^{k} |x_i|^2$.

- A pyramid is a set of the form $\{x : \sum_{i=1}^{k} |x_i| \leq c\}$.
  (sphere = $\{x : \sum_{i=1}^{k} |x_i|^2 \leq c\}$)

- Intended for densities such as IID Laplacian, which is constant on surface of a pyramid:
  $$p(x) = 2^{-k/2} e^{-\sqrt{2} \sum_{i=1}^{k} |x_i|}$$

- There are low complexity partitioning algorithms.
**Transform Coding (Quantization)**

- **Motivation:** How to make scalar quantizers work well for correlated source, e.g., Gaussian AR source, $\rho \approx 1$:
  - $k$ scalar quantizers form a Direct Product Code, with spherical or elliptical point density.
  - The transform rotates the source so as to align its probability density with the product point density. Notice the asymmetry.
  - Resulting $k$-dimensional VQ has point density lined up with source density. (What about cell shapes?)

- We will carefully analyze transform codes.

\[ X \xrightarrow{T} \begin{array}{l}
U_1 \\
U_2 \\
\vdots \\
U_k \\
\end{array} \xrightarrow{Q} \\
\begin{array}{l}
V_1 \\
V_2 \\
\vdots \\
V_k \\
\end{array} \xrightarrow{T^{-1}} Y \\
\]

$k$ Scalar Quantizers = Product Quantizer
• Transform codes form the basis of most image and video coders, and some audio coders.

• Choice of transform and choice of coefficient coding technique are big issues.

• Common transforms: DFT, DCT, wavelet (many kinds)

• Coefficient coding technique: uniform scalar quantization with entropy coding is common (e.g. JPEG, JPEG 2000).
  Choices for the quantizer step sizes and for how to organize the quantized coefficients for joint or conditional coding is big issue.
  Vector quantization or DPCM can also be used.

• Subband Coding: A closely related technique. Commonly used for image, video and audio coding.
  Apply transform to blocks.
  Group like coefficients from different blocks into "bands".
  To each band, apply a lossy coder designed specifically for that band.
  Sometimes the bands are created by bandpass filtering the data sequence, without parsing the data into k-dimensional blocks.

![DPCM: Differential Pulse Code Modulation](image)

• Motivation: How to make a scalar quantizer work for a source with memory?
  \[ Y_i = \bar{X}_{i-1} + Q(X_i-Y_{i-1}) = \text{prediction} + \text{quantized prediction error} \]
  \[ X_i-Y_i = U_i - Q(U_i), \ i.e. \ \text{overall error} = \text{scalar quantizer error'} \]
  \[ D = \text{distortion of scalar quantizer}, \ R = \text{rate of scalar quantizer}. \]
  Binary encoder can be fixed or variable rate.
  The predictor permits the scalar quantizer to be applied to something with small variance, so that it and the resulting DPCM coder have good D vs. R.
  Can view this as a kind of backward adaptive quantization. The prediction has a shifting affect on the quantizer.
  The term "pulse code modulation" has lost its original significance.
  Issues: Choice of predictor, type of binary encoder, design of quantizer.
  Often the predictor and quantizer are "adaptive".
  DPCM is commonly used in speech coding (older methods). Also used in most video coding methods (interframe coding).