EE5585 Data Compression March 14, 2013

Lecture 15

Instructor: Arya Mazumdar *Scribe: Khem Chapagain*

Scalar Quantization Recap

Quantization is one of the simplest and most general ideas in lossy compression. In many lossy compression applications, we are required to represent each source output using one of a small number of codewords. The number of possible distinct source output values is generally much larger than the number of codewords available to represent them. The process of representing a large - possibly infinite - set of values with a much smaller set is called quantization.

In the previous lectures, we looked at the uniform and nonuniform scalar quantization and moved on to vector quantizers. One thing that's worth mentioning about scalar quanitization is that, for high rate, uniform quantizers are optimal to use. That can be seen from the following derivation. As its name implies, scalar quantizer inputs are scalar values (each input symbol is treated separately in producing the output), and each quantizer output (codeword) represents a single sample of the source output.

- R.V. to quantize: X,
- Set of (decision) boundaries: $\{b_i\}_{i=0}^{M}$,
- Set of reconstruction/quantization levels/points: $\{y_i\}_{i=1}^{M}$, i.e., if x is between b_{i-1} and b_i, then it's assigned the value of y_i.

The Optimal Quantizer has following two properties:

1. $y_i = \dfrac{\int_{b_{i-1}}^{b_i} x\, f_X(x)\, \mathrm{d}x}{\int_{b_{i-1}}^{b_i} f_X(x)\, \mathrm{d}x}$
2. $b_i = \dfrac{y_i \; + \; y_{i+1}}{2}$

If we want a high rate quantizer, the number of samples or quantization levels (M) is very large. Consequently, difference between boundaries b_{i-1} and b_i is very small as they are very closely spaced.

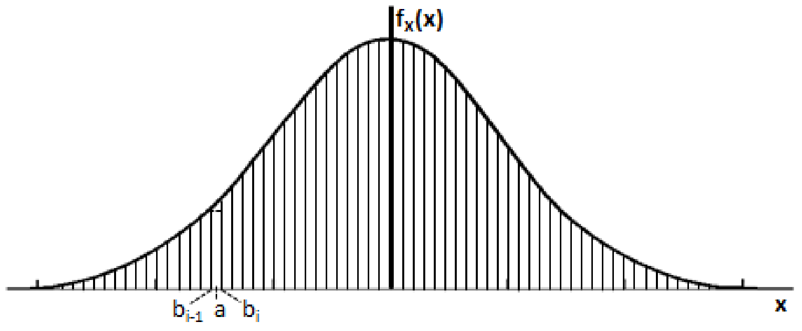

Figure 1: A Gaussian *pdf* with large number of quantization levels (M).

When M is sufficiently large, probability density function (*pdf*), $f_X(x)$, doesn't change much between b_{i-1} and b_i. In that case, the first property above can be written as,

$$\begin{aligned} y_i &\approx \frac{f_X(a) \int\limits_{b_{i-1}}^{b_i} x \, \mathrm{d}x}{f_X(a) \int\limits_{b_{i-1}}^{b_i} \mathrm{d}x}, \quad b_{i-1} \le a \le b_i \\ &= \frac{(b_i^2 - b_{i-1}^2)/2}{(b_i - b_{i-1})} \\ &= \frac{b_i + b_{i-1}}{2} \end{aligned}$$

Which says that for very large M, reconstruction level is approximately in the midway between two neighboring boundaries, but we know from the second property above that boundary is exactly between two neighboring reconstruction levels. This means that we have a uniform quantizer since it has all reconstruction levels equally spaced from one another and quantization steps (intervals) are equal. Thus, when we are operating with a large M, it makes sense just to go for the uniform quantizers.

Lloyd Algorithm

In the last class, we talked about Lloyd-Max algorithm to iteratively compute the optimized quantizer levels (y_i's) and boundaries (b_i's). There is another algorithm called Lloyd algorithm, which also iteratively calculates y_i's and b_i's, but in a different way. The Lloyd quantizer that is constructed using an iterative procedure provides the minimum distortion for a given number of reconstruction levels, in other words, it generates the *pdf*-optimized scalar quantizer. The Lloyd algorithm functions as follows (source distribution $f_X(x)$ is assumed to be known):

- Initialization:
  - Assume an initial set of reconstruction values/levels, $\{y_i^{(0)}\}_{i=1}^{M}$, where $^{(0)}$ denotes the 0^{th} iteration.
  - Find decision boundaries, $\{b_i^{(0)}\}_{i=0}^{M}$ from the equation $b_i^{(0)} = \dfrac{y_i^{(0)} + y_{i+1}^{(0)}}{2}$
  - Compute the distortion (variance), $D^{(0)} = \displaystyle\sum_{i=1}^{M} \int\limits_{b_{i-1}^{(0)}}^{b_i^{(0)}} (x - y_i^{(0)})^2 \, f_X(x) \, \mathrm{d}x$
- Update rule: do for $k = 0, 1, 2, ...$
  - $y_i^{(k+1)} = \dfrac{\int\limits_{b_{i-1}^{(k)}}^{b_i^{(k)}} x \, f_X(x) \, \mathrm{d}x}{\int\limits_{b_{i-1}^{(k)}}^{b_i^{(k)}} f_X(x) \, \mathrm{d}x}$, new set of reconstruction levels.
  - $b_i^{(k+1)} = \dfrac{y_i^{(k+1)} + y_{i+1}^{(k+1)}}{2}$, new set of boundaries.

- $D^{(k+1)} = \sum_{i=1}^{M} \int_{b_{i-1}^{(k+1)}}^{b_i^{(k+1)}} (x - y_i^{(k+1)})^2 \, f_X(x) \, \mathrm{d}x$, new distortion.

- Stop iteration if $|D^{(k+1)} - D^{(k)}| < \epsilon$, where ϵ is a small tolerance > 0; i.e., stop when distortion is not changing much anymore.

We will see later that this algorithm can be generalized to vector quantization.

Vector Quantization

The set of inputs and outputs of a quantizer can be scalars or vectors. If they are scalars, we call the quantizers scalar quantizers. If they are vectors, we call the quantizers vector quantizers. By grouping source outputs together and encoding them as a single block, we can obtain efficient compression algorithms. Many of the lossless compression algorithms take advantage of this fact such as clubbing symbols together or parsing longest phrases. We can do the same with quantization. We will look at the quantization techniques that operate on blocks of data. We can view these blocks as vectors, hence the name "vector quantization." For a given rate (in bits per sample), use of vector quantization results in a lower distortion than when scalar quantization is used at the same rate. If the source output is correlated, vectors of source output values will tend to fall in clusters. By selecting the quantizer output points to lie in these clusters, we have a more accurate representation of the source output.

We have already seen in the prior class that, if we have the data that are correlated, it doesn't make sense to do scalar quantization. We looked at the height(H), weight(W) quantization scheme in two dimensions. We have two random variables (H, W) and will plot the height values along the x-axis and the weight values along the y-axis. In this particular example, the height values are being uniformly quantized to the five different scalar values, as are the weight values. So, we have a total of 25 quantization levels ($M = 25$) denoted by •. The two-dimensional representation of these two quantizers is shown below.

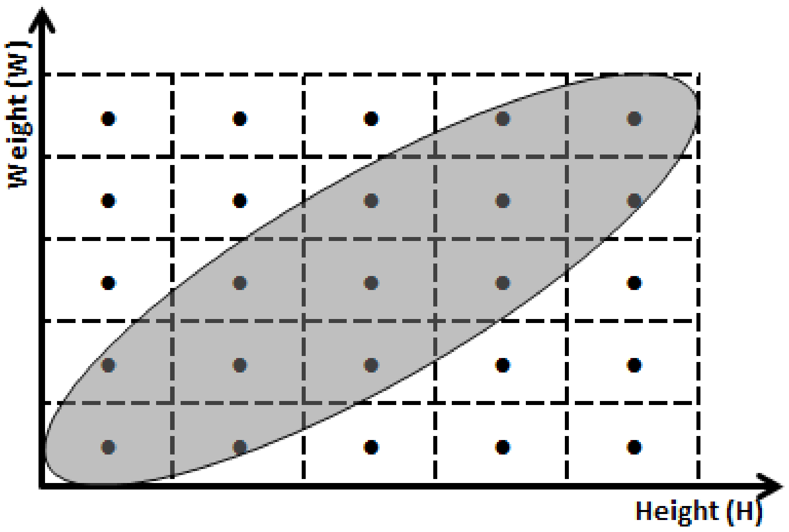

Figure 2: The height, weight scalar quantizers when viewed in two dimensions.

From the figure, we can see that we effectively have a quantizer output for a person who is very tall and weighs minimal, as well as a quantizer output for an individual who is very short but weighs a lot. Obviously, these outputs will never be used, as is the case for many of the other unnatural outputs. A more sensible approach would be to use a quantizer with many reconstruction points inside the shaded region shown in the figure, where we take account of the fact that the height and weight are correlated. This quantizer will have almost all quantization levels (say 23) within this shaded area and no or very few (say 2) quantization levels outside of it. With this approach, the output points are clustered in the area occupied by the most likely inputs. Using this methodology provides a much finer quantization of the input. However, we can no longer quantize the height and weight separately - we have to consider them as the coordinates of a point in two dimensions in order to find the closest quantizer output point.

When data are correlated, we don't have any doubt that quantizing data together (vector quantization) is going to buy us something. It will be shown, when data are independent (no correlation), we still gain from using vector quantization. Scalar quantization always gives something like a rectangular grid similar to what we have seen in the above example. Note that for *pdf*-optimized or optimal quantizer, unlike above example, grids/cells can be of different sizes meaning boundaries don't necessarily have to be uniformly spaced in either dimension. Nevertheless, the fact is that these rectangles don't fill up the space most efficiently. There might be other shapes, like a hexagon or something else, that can fill up the space in a better way with the same number of quantization levels. This idea is captured in vector quantization, where we have this flexibility of using any shape, contrasting the scalar case where only choice is rectangle like shapes.

Exercise: A square and a regular hexagon having equal area.

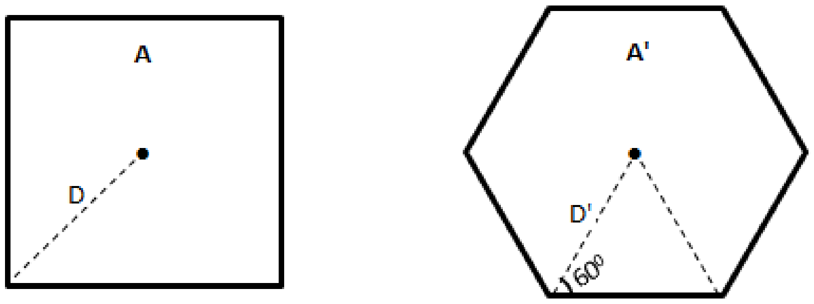

Figure 3: A square and a regular hexagon of same area.

Area of the square,

$$\begin{aligned} A &= (\sqrt{2}D)^2 \\ &= 2D^2 \end{aligned}$$

Area of the hexagon is six times the area of the inside triangle shown,

$$\begin{aligned} A' &= 6\, D' \cos(\frac{\pi}{3})\, D' \sin(\frac{\pi}{3}) \\ &= 6\, \frac{\sqrt{3}}{4} D'^2 \\ &= \frac{3\sqrt{3}}{2} D'^2 \end{aligned}$$

If the areas are equal,

$$
\begin{aligned}
A' &= A \\
\frac{3\sqrt{3}}{2}D'^2 &= 2D^2 \\
D'^2 &= \frac{4}{3\sqrt{3}}D^2 \\
D'^2 &= \frac{4}{5.196}D^2 \\
D' &= 0.877\,D \\
D' &< D
\end{aligned}
$$

Even though, both the square and hexagon occupy the same area, the worst case distortion in hexagon (D') is less than the worst case distortion in square (D). Hence, we can intuitively see that hexagon can fill up the same space with less distortion (better packing), which is also known as the shape gain. This is illustrated in the following figure.

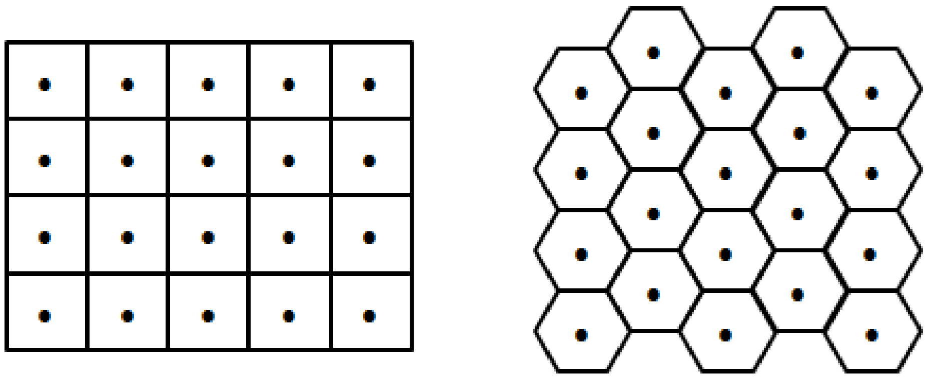

Figure 4: 20 square cells and 20 hexagonal cells packing the equal area.

From earlier discussions, it is clear that for correlated data, vector quantizers are better. We just saw that, even if the the data are not correlated, vector quantizers still offer this shape gain among other advantages. Without having all this intuition and geometry, this fact was captured previously in the rate distortion theory as well.

We can try to extend Lloyd algorithm to vector quantizers. Now, we have a space to quantize instead of a real line. Analogous to scalar case this time we have,

- Given r.v.'s (need not be iid), X_1, X_2, ..., X_n, $\quad X_i \in \mathbb{R}$
- Joint *pdf*, $f_{X_1,\ldots,X_n}(x_1,\, \ldots,\, x_n)$ determines the range (range doesn't always have to be $\mathbb{R}$).
- $\{\underline{Y_i}\}_{i=1}^{M}$, set of quantization levels (n dimensional vectors).
- $\{V_i\}_{i=1}^{M}$, boundaries are hyper-planes or partitions of $\mathbb{R}^n$, i.e., $\displaystyle\bigcup_{i=1}^{M} V_i = \mathbb{R}^n$

Given a set of quantization/reconstruction levels/points, we want find the optimal partitions. Suppose quantization levels are,

$$\{\underline{Y_i}\}_{i=1}^{M}$$

V_i, the Voronoi region of $\underline{Y_i}$, is the region such that any point in V_i is closer to $\underline{Y_i}$ than any other $\underline{Y_j}$.

$$V_i = \{\underline{X} = (x_1,\, ...,\, x_n) \in \mathbb{R}^n \mid d(\underline{Y_i},\, \underline{X}) < d(\underline{Y_j},\, \underline{X})\ \forall\ j \neq i\}$$

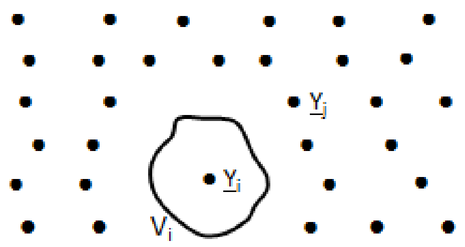

Figure 5: V_i is the Voronoi region of $\underline{Y_i}$.

And the distortion is,

$$\begin{aligned} D &= \int\limits_{\mathbb{R}^n} \|\underline{X} - Q(\underline{X})\|_2^2\ f_{\underline{X}}(\underline{X})\ \mathrm{d}\underline{X} \\ &= \sum_{i=1}^{M} \int\limits_{V_i} \|\underline{X} - \underline{Y_i}\|_2^2\ f_{\underline{X}}(\underline{X})\ \mathrm{d}\underline{X} \\ &= \sum_{i=1}^{M} \int\limits_{V_i} (\underline{X} - \underline{Y_i})^T (\underline{X} - \underline{Y_i})\ f_{\underline{X}}(\underline{X})\ \mathrm{d}\underline{X} \end{aligned}$$

Where $d(.)$ is some distance metric, $Q(.)$ is the quantizer function that maps any value in V_i to $\underline{Y_i}$, and $\|.\|_2^2$ is the Euclidean norm (we can use other norms also).

Linde-Buzo-Gray (LBG) Algorithm

Linde, Buzo, and Gray generalized Lloyd algorithm to the case where the inputs are no longer scalars. It is popularly known as the Linde-Buzo-Gray or LBG algorithm, or the generalized Lloyd algorithm. For the case where the distribution is known, the algorithm looks very much like the Lloyd algorithm described earlier for the scalar case.

- Initialization:
  - Assume an initial set of quantization/reconstruction points/values, $\{\underline{Y_i}^{(0)}\}_{i=1}^{M}$
  - Find quantization/reconstruction/voronoi regions,
    $V_i^{(0)} = \{\underline{X}\ :\ d(\underline{X},\, \underline{Y_i}^{(0)}) < d(\underline{X},\, \underline{Y_j}^{(0)})\ \forall\ j \neq i\}$
  - Compute the distortion, $D^{(0)} = \sum_{i=1}^{M} \int\limits_{V_i^{(0)}} (\underline{X} - \underline{Y_i}^{(0)})^T (\underline{X} - \underline{Y_i}^{(0)})\ f_{\underline{X}}(\underline{X})\ \mathrm{d}\underline{X}$
- Update rule: do for $k = 0,\, 1,\, 2,\, ...$
  - $\underline{Y_i}^{(k+1)} = \dfrac{\int_{V_i^{(k)}} \underline{X}\ f_{\underline{X}}(\underline{X})\ \mathrm{d}\underline{X}}{\int_{V_i^{(k)}} f_{\underline{X}}(\underline{X})\ \mathrm{d}\underline{X}}$, this represents the point (centroid) that minimizes distortion within that region.

  - $V_i^{(k+1)} = \{\underline{X} \; : \; d(\underline{X}, \underline{Y_i}^{(k+1)}) < d(\underline{X}, \underline{Y_j}^{(k+1)}) \; \forall \; j \neq i\}$, new set of reconstruction regions.
  - $D^{(k+1)} = \displaystyle\sum_{i=1}^{M} \int_{V_i^{(k+1)}} (\underline{X} - \underline{Y_i}^{(k+1)})^T (\underline{X} - \underline{Y_i}^{(k+1)}) \; f_{\underline{X}}(\underline{X}) \; \mathrm{d}\underline{X}$, updated distortion.

- Stop when $|D^{(k+1)} \; - \; D^{(k)}| < \epsilon$, where ϵ is a small tolerance > 0.

This algorithm is not very practical because the integrals required to compute the distortions and centroids are over odd-shaped regions (hyper-space) in n dimensions, where n is the dimension of the input vectors. Generally, these integrals are extremely difficult to compute and trying every vector to decide whether it's in the Voronoi region or not is also unfeasible, making this particular algorithm more of an academic interest. Of more practical interest is the algorithm for the case where we have a training set available. In this case, the algorithm looks very much like the k-means algorithm.

- Initialization:
  - Start with a large set of training vectors, $\underline{X_1}, \underline{X_2}, \underline{X_3}, ..., \underline{X_L}$ from the same source.
  - Assume an initial set of reconstruction values, $\{\underline{Y_i}^{(0)}\}_{i=1}^{M}$
  - Find quantization/voronoi regions, $V_i^{(0)} = \{\underline{X_i} \; : \; d(\underline{X_i}, \underline{Y_i}^{(0)}) < d(\underline{X_i}, \underline{Y_j}^{(0)}) \; \forall \; j \neq i\}$, here we need to partition only L vectors instead of doing it for all vectors.
  - Compute the distortion, $D^{(0)} = \displaystyle\sum_{i=1}^{M} \sum_{\underline{X} \in V_i^{(0)}} (\underline{X} - \underline{Y_i}^{(0)})^T (\underline{X} - \underline{Y_i}^{(0)})$

- Update rule: for $k = 0, 1, 2, ...$
  - $\underline{Y_i}^{(k+1)} = \dfrac{1}{|V_i^{(k)}|} \displaystyle\sum_{\underline{X} \in V_i^{(k)}} \underline{X}$, find the centroid (center of mass of points), here $|.|$ is the *size of* operator.
  - $V_i^{(k+1)} = \{\underline{X_i} \; : \; d(\underline{X_i}, \underline{Y_i}^{(k+1)}) < d(\underline{X_i}, \underline{Y_j}^{(k+1)}) \; \forall \; j \neq i\}$, new set of decision regions.
  - $D^{(k+1)} = \displaystyle\sum_{i=1}^{M} \sum_{\underline{X} \in V_i^{(k+1)}} (\underline{X} - \underline{Y_i}^{(k+1)})^T (\underline{X} - \underline{Y_i}^{(k+1)})$, new distortion.

- Stop when $|D^{(k+1)} \; - \; D^{(k)}| < \epsilon$, where ϵ is a small tolerance > 0.

We didn't know the *pdf* of the data or source, all we had was a set of training data. Yet, when this algorithm stops, we have a set of reconstruction levels ($\underline{Y_i}$'s) and quantization regions (V_i's), which gives us a vector quantizer. This algorithm is a more practical version of the LBG algorithm. Although, this algorithm forms the basis of most vector quantizer designs, there are few issues with this algorithm, for example,

1. **Initializing the LBG Algorithm:** What would be the good set of initial quantization points that will guarantee the convergence? The LBG algorithm guarantees that the distortion from one iteration to the next will not increase. However, there is no guarantee that the procedure will converge to the optimal solution. The solution to which the algorithm converges is heavily dependent on the initial conditions and by picking different subsets of the input as our initial codebook (quantization points), we can generate different vector quantizers.

2. **The Empty Cell Problem:** How do we take care of a situation when one of the reconstruction/quantization regions in some iteration is empty? There might be no points which are closer to a given reconstruction point than any other reconstruction points. This is a problem because in order to update an output point (centroid), we need to take the average value of the input vectors.

Obviously, some strategy is needed to deal with these circumstances. This will be the topic of next class after the Spring Break.

Practical version of LBG algorithm described above is surprisingly similar to the k-means algorithm used in data clustering. Most popular approach to designing vector quantizers is a clustering procedure known as the k-means algorithm, which was developed for pattern recognition applications. The k-means algorithm functions as follows: Given a large set of output vectors from the source, known as the training set, and an initial set of k representative patterns, assign each element of the training set to the closest representative pattern. After an element is assigned, the representative pattern is updated by computing the centroid of the training set vectors assigned to it. When the assignment process is complete, we will have k groups of vectors clustered around each of the output points (codewords).

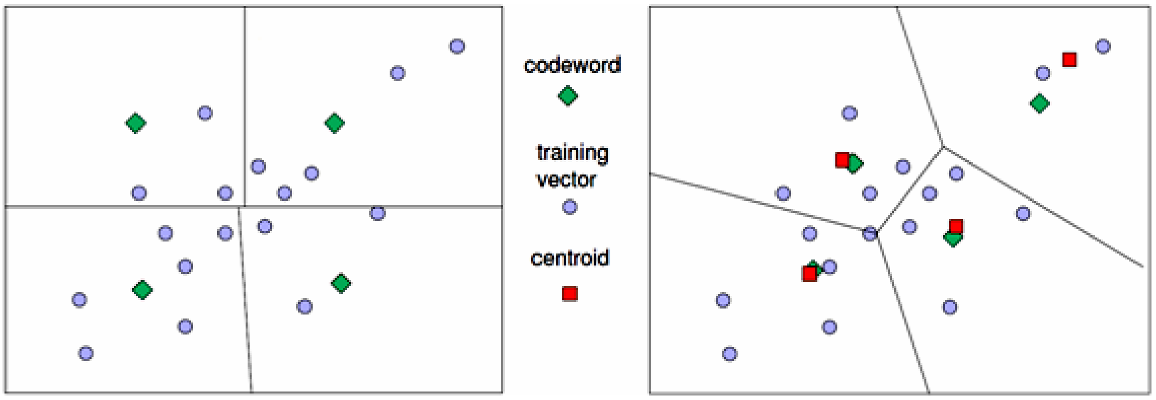

Figure 6: Initial state (left) and final state (right) of a vector quantizer.

We have seen briefly how we can make use of the structure exhibited by groups, or vectors, of values to obtain compression. Since there are different kinds of structure in different kinds of data, there are a number of different ways to design vector quantizers. Because data from many sources, when viewed as vectors, tend to form clusters, we can design quantizers that essentially consist of representations of these clusters.