ILLINOIS INSTITUTE OF TECHNOLOGY
ECE 508 SIGNAL AND DATA COMPRESSION (Fall 1999)
PROJECT #1
Scalar and Vector Quantizer Design for Quantization of Speech Samples

Due Date: November 1, 1999

Objectives

The major goals of this computer simulation project are:

1. Implement the generalized Lloyd algorithm (GLA) to enable the design of scalar and vector quantizer codebooks from training data.
2. Design scalar and vector codebooks for quantization of speech samples.
3. Evaluate the performance of the designs.

Ground Rules

You must write your own programs, run your own simulations, and analyse and interpret your simulation results. Submit a report **on paper**. Document any theoretical/analytical preparation you did (or simply refer to specific material in the textbook) as part of your design of your computer programs. Document your simulation results and plots, any extra effort in addition to the required work detailed below, and your observations and conclusions. Grading is based on the quality of your report and simulation results and any "extra mile" you put into the work. Present your results compactly, clearly, and selectively. A report "stuffed" with redundant material will not qualify for a high grade. Email all the programs you have written for the project, in one concatenated **plain-text** file, to chan@ece.iit.edu. Do not include your computer programs with your report.

You may discuss with your classmates about the project, but your effort must be *essentially independent.* Verification of independent effort will be performed, and plagiarism will be treated as academic dishonesty. Developing your own learning and judgement skills is more rewarding than recycling others'.

You may, if you wish, use "canned" library functions to calibrate your simulation results. Comparing results from two different computer programs that purport to perform the same task is not always trivial; so it might not be worth your time. Nevertheless, if you choose to do so, document it in your report. Otherwise, library functions should only be used to analyse and visualize your simulation results.

Include a comment header with your program describing on what machine the program was compiled and run (e.g. gcc on Sun Unix, Borland C on Windows 95). For each program module, include at least a brief comment statement describing the function of the module. This course is not about programming, so you are free to choose any computer language you like. Any reasonable implementation will be acceptable.

Implementing the GLA

Practical issues that need to be addressed in implementing the GLA are:

1. Choice of the initial codebook. There are many ways to initialize the codebook. For instance,

   - generate a uniform or lattice quantizer codebook
   - pick samples from the training set using some selection method such as random sampling, sampling with a minimum distance admission criterion, etc
   - "splitting" or "tree-growing"

   Running the GLA with different initializations will lead to different codebooks of different quality.

2. Detection and correction of "empty cells." When upon completion of the encoder optimization step of the GLA, one or more partition regions/cells are found to have not "attracted" a single training vector from the training set, then centroids can not be computed for these empty cells. Empty cells can be corrected by assigning a portion of the training vectors of the heavily populated cells to the empty cells. To do so, one needs to maintain population counts for the cells, and devise a selection and sharing procedure that would handle more than one empty cell.

   Note that a "nearly empty" cell may also be problematic. On the other hand, it is to be expected (why?) that some cells will have a larger proportion of the training population than other cells. Moreover, a speech file that has many "silence" intervals and a relatively stationary background noise could result in some cells with very large counts.

   Any empty cell correction procedure that you devise is likely to violate the optimality conditions undergirding the GLA. What implication does that have for the monotonic improvement property of the GLA, and the convergence time of the algorithm?

3. VQ codebook design can be very computation intensive. There are two sources of complexity growth: (i) GLA iterates between optimizing the encoder and the decoder, but encoder complexity grows exponentially with the code rate and the vector demension; (ii) the training set size is usually chosen to be some fixed multiple (the "training ratio") of the size of the codebook, but codebook size also grows exponentially with the code rate and the vector demension. Therefore it is prudent to write the compute-intensive part (the "computational kernel") of your GLA program as efficiently as possible. If your computer has lots of RAM, try keeping as much of the data in RAM as possible. Avoid unnecessary indexing and movement of data, and do some algebra by hand to minimize the amount of computation before commiting to programming. Some of the computations may serve multiple purposes, so that you can avoid repeating them, or you can reuse intermediate results. Of course, the saving would only be significant for calculations that represent a significant fraction of the total amount of computations. And if run time and computer access is not a problem for you, maybe it is not worthwhile to spend a lot of time optimizing your code.

Depending on the speed of your computer, it could take hours to design a large codebook. Using an interpreter based computer language such as Matlab to design large VQ codebooks is not recommended. You can project the run time for designing a large codebook by first designing a smaller codebook and note its run time and number of GLA iterations. (In any case, to avoid wasting time on generating garbage results, it would be prudent to test and debug your program on a small data set first.) Once you know the run time per GLA iteration for a given codebook size and data set size, you can then project the run time for a different codebook and training set size. The number of GLA iterations needed to achieve "near convergence" depends on a number of factors, such as how close is the initial codebook to some "stationary point" on the error (MSE) surface. Typical number of iterations is between 10 and 20. A smaller number is not necessarily an indication of poor codebook quality.

You might find the design examples described in Section 11.4 of Gersho and Gray helpful to calibrate your results. Note however that the rate-distortion characteristic for real-life sources such as speech can vary from one data set to the next, depending on a host of factors, e.g. how the signal was filtered, the number of speakers, the language, etc. Since you do not have access to the original data used to generate the results shown in Section 11.4, you should not try to match the results on an absolute basis.

To ease debugging and to monitor the "health" of your program and your simulation runs, you should consider having your program write out vital statistics to a journal file as simulation progresses. For instance, you can monitor MSE convergence, cell population statistics, etc. For very long simulation runs, you may even file away a copy of the codebook obtained in each GLA iteration. If the program crashes, you can restart from the latest codebook.

Designing Codebooks for Quantizing Speech Samples

Speech files are available on the course Web site. See the appropriate README files for guidance. The larger speech file contains the training set and the smaller the test set. Note that Unix files store the most significant byte first, whereas PC files store the least significant byte first. (Actually, byte ordering is determined by the machine architecture rather than the operating system.) Byte swapping might be necessary, depending on your target machine and file download method.

Apply the *original* training set to your GLA program to design pdf-optimized codebooks for various values of vector dimension k and code rate r in bits per sample. Take every k samples from the file to form a training vector; do not reuse any sample. Suggested values of k are 1, 2, 4, 8. Maintain a training ratio of no less than 50. This constraint and the size of the training set (and perhaps program run time) will determine the maximum r you can attempt.

Also, design *uniform* scalar quantizers for different values of r for the training set. Describe how you chose the step size as a function of r, using the training set? Suggested values of r are: 1, 2, 4, 8, 12.

Performance Evaluation

Evaluate the rate-distortion performance of the quantizers you have designed. "Rate" is measured using "code rate" in bits per sample; actual bit rate depends on how the quantizer output points are coded into binary bits. "Distortion" is measured using mean-squared error (MSE), where the averaging is taken over the entire data set. In order to compare results between different data sets, MSE distortion should be expressed in terms of signal-to-noise ratio (SNR) in decibels.

Apply your designed quantizers to produce quantized speech, separately from the training set and from the test set. Calculate separate SNR (dB) values for the training set and the test set. Plot SNR (dB) performance as a function of k and r. Organize the data in different plots so you can make appropriate observations and comparisons. As an additional data point for your performance graphs, you may include the SNR of the μ-law quantized speech on the course Web page. Consider the following issues:

1. Correctness check: do the performance differences between different types (uniform scalar, Lloyd-Max, VQ with a given k) of quantizers make sense? For instance, which quantizer gives the best performance for a given r? Is there any analmoly that might suggest the presence of flaws in your experiments?

2. Can you explain the performance difference between the various types of quantizers, for a given r? Can you render your explanation quantitatively, making use of high-resolution theoretical results?

3. Can you explain the performance trend as a function of r, for each type of quantizer? Is there any trend that matches theoretical prediction?

4. Is there any great descrepancy between the training and the test SNRs? If so, what are the possible causes?

You may optionally compare the quality of quantized speech files by listening to them. If you choose to do so, you would need to use a headphone and listen carefully in a relatively quiet room. Tools for playing speech files are listed on the course Web page. Listen for noise and distortions, and note any change in quality and naturalness. Try ranking the speech quality between the different quantized files without looking over the SNR results. Does your subjective quality ranking match well with SNR ranking? What implication can you draw regarding using SNR as a predictor of the subjective quality of quantized speech?

As a novelty, you may subtract the quantized speech from the original speech to obtain the quantization error signal. Listen to the error signals produced by using different r and k values. What kind of sound can you hear? Is there any telltale sign in the error signal about the original speech signal? How does the texture of the error signal vary with r and k? Does your listening suggest any sort of model that may be used to relate the error signal to the original signal and the quantizer (see, for example, the latter part of Section 5.6 of the textbook)?

Another novelty: some Matlab licenses include a voronoi function which you can use to plot the Voronoi partition of a two-dimensional vector quantizer. You can use the command to observe the evolution of the partition with the GLA iterations.