Conjugate Gradient Methods For Designing Vector Quantizers

Eyal Yair †, Kenneth Zeger, and Allen Gersho

Center for Information Processing Research
Department of Electrical & Computer Engineering
University of California, Santa Barbara, CA 93106

Abstract

Vector quantizer design procedures attempt to minimize a multi-dimensional cost function. Generally, the global minimum is too difficult to find and local minima are sought using descent algorithms for which the cost function decreases monotonically. One of the most common techniques today for VQ design is the Generalized Lloyd Algorithm (GLA), a descent algorithm which is convenient to use but for which no results on its computational efficiency have been established. This paper introduces a new technique for efficient VQ design, based on conjugate gradient search methods. It is demonstrated experimentally that the the new design algorithm performs better than the GLA in many cases in terms of computation time and/or quality of the codebook it produces.

1. Introduction

Many descent algorithms are known to locally optimize a multi-dimensional cost function (e.g., [1]) but there is no conclusive algorithm which can be said to outperform all the other methods for a non-quadratic cost function. The Generalized Lloyd Algorithm (GLA) [2], also known as the LBG algorithm, is a method specifically developed for the problem of quantizer design. It is a descent algorithm for which the cost function decreases monotonically. There is, however, no evidence that the GLA is superior to other optimization techniques which are not necessarily specific to the VQ design problem.

Two alternatives to the GLA for minimizing the cost function, namely the *Steepest Descent* (SD) method and the *Conjugate Gradient* (CG) method, are studied in this paper and their performance is compared to the GLA. Although these techniques are widely used in other optimization problems, so far they have not been used for VQ design. We first show how the SD method can be applied to VQ design and demonstrate that the GLA is similar to the SD method but the codevectors in the GLA are not necessarily updated in the direction of the negative gradient. We then introduce the application of the CG method to the non-quadratic case of VQ design and compare its performance with the GLA.

In the SD method the codevectors are updated at each iteration in the direction of the negative gradient of the cost function, using an optimal step-size obtained (analytically) by a line minimization along the gradient direction. Is is well-known that the speed of convergence of such a technique is bounded by the condition number of the Hessian matrix (given by the ratio of its largest and smallest eigenvalues) [1].

The CG algorithm is known to generally outperform the steepest descent method for many practical applications with only marginally additional computation requirements. Instead of choosing a fixed direction in the codebook space, as with the negative gradient in the SD method, the direction in which the codevectors are updated is determined dynamically based on the current value of the gradient and the previous update direction. For a quadratic cost function, the conjugate gradient algorithm is known to converge in a finite number of steps, and the upgraded speed of convergence with respect to the SD method is achieved by effectively eliminating the influence of the largest eigenvalues of the Hessian matrix. In the case of a non-quadratic cost function a *partial conjugate gradient* (PCG) algorithm is often employed, in which the update direction is periodically reset to the negative gradient direction, and conjugate gradient directions are used between two restarts. The PCG algorithm thus includes the SD method as a special case when the restart is performed every iteration.

In this paper, the application of both the SD and the PCG methods to vector quantizer design are introduced. The computational requirements per iteration for both algorithms are essentially the same as in the GLA. Results comparing these techniques with the GLA for Gauss-Markov and speech sources are given, demonstrating the superiority of the PCG algorithm over the GLA in terms of designing a quantizer with an improved trade-off between SNR achieved and computation time.

2. Steepest Descent Method and the GLA

Denote the VQ codevectors by $\mathbf{y}_i$ $(i = 1, \cdots, N)$, the codebook by Y, the cost function by D, and the set of training vectors by $\boldsymbol{T}$. Usually a vector quantizer is defined by jointly specifying its partition cells and its codebook. An alternative viewpoint, which we follow here, is to define a quantizer solely by specifying its codebook with the implicit requirement that the partition cells are *always* the nearest neighbor cells for the given codebook. Thus, a quantizer is a mapping of an input vector $\mathbf{x}$ into that codevector in the codebook Y to which it is nearest in the Euclidean distance sense. Hence, the average dis-

This work was supported by the Weizmann Foundation for scientific research, the University of California MICRO program, Bell Communications Research, Bell-Northern Research, and Rockwell International.

† Author is currently at the IBM Scientific Center in Haifa, Israel.

CH2847-2/90/0000-0245 $1.00 © 1990 IEEE

tortion of a quantizer is a function only of the codebook Y, for a given input source distribution. Based on the mean-squared quantization error as a performance measure and taking the source distribution as being specified by the training set $\boldsymbol{T}$, the cost function to be minimized by a design algorithm is given by

$$D(Y) \triangleq \tfrac{1}{2} \sum_{i=1}^{N} \sum_{\mathbf{x} \in R_i} \| \mathbf{x} - \mathbf{y}_i \|^2 \quad , \tag{2.1}$$

where each R_i is the nearest-neighbor cell of the codevector $\mathbf{y}_i$, given by

$$R_i = \{ \mathbf{x} \in \boldsymbol{T} : \| \mathbf{x} - \mathbf{y}_i \| \leq \| \mathbf{x} - \mathbf{y}_j \| \text{ for all } j \neq i \} \tag{2.2}$$

where each $\mathbf{x} \in R_i$ is quantized to the codevector $\mathbf{y}_i$. Usually an arbitrary tie-breaking rule is assumed so that a training vector $\mathbf{x}$ on the boundary between two (or more) nearest-neighbor cells is uniquely assigned to one of these cells. However, for any given codebook, if a training set is generated from a continuously distributed source, the probability is zero that any training vectors would lie on the boundary.

It should be noted that the quantity $D(Y)$ depends both directly on the codevectors $\mathbf{y}_i \in Y$ arising in each term of the inner summation in (2.1) and indirectly on the range of the inner summation since each region R_i is implicitly a function of the codebook as given by (2.2). The goal of the design is to find a codebook Y that locally minimizes $D(Y)$. The GLA is an iterative procedure in which, at each step, (a) the codevectors are replaced by the centroids of their nearest-neighbor cells and then (b) a new set of nearest neighbor cells is determined by the new codevectors. With our definition of a quantizer, the second step, (b), is automatically implied and need not be specifically stated. Therefore in the GLA, at each iteration the codevectors are updated according to

$$\mathbf{y}_i \leftarrow \mathbf{c}_i \triangleq \frac{1}{K_i} \sum_{\mathbf{x} \in R_i} \mathbf{x} \quad , \tag{2.3}$$

where $\mathbf{c}_i$, called the centroid of R_i, is the sample mean of the training vectors in R_i, and K_i is the number of training vectors in R_i.

In order to compute the gradient of the cost function, we make the *zero-boundary* assumption that for the current codebook Y, none of the training vectors $\mathbf{x} \in \boldsymbol{T}$ lie on the boundary between two nearest-neighbor regions. From (2.1) it can be seen that the partial derivatives of the cost function $D(Y)$ with respect to the codevectors, denoted by $\mathbf{g}_i$, are given by

$$\mathbf{g}_i \triangleq \frac{\partial D}{\partial \mathbf{y}_i} = \sum_{\mathbf{x} \in R_i} (\mathbf{y}_i - \mathbf{x}) = K_i (\mathbf{y}_i - \mathbf{c}_i) \tag{2.4}$$

for $i = 1, \cdots, N$. Note that the sets R_i are also functions of the codevectors. However, since the training set $\boldsymbol{T}$ is finite, under the zero-boundary assumption infinitesimally small changes in $\mathbf{y}_i$ do not change the sets R_i and these sets can be treated as constants with respect to $\mathbf{y}_i$ in the derivative calculation. While we have no theoretical guarantee that for a fixed training set, the zero-boundary condition will hold after an arbitrary algorithmic change to the codebook, in the rare and improbable event that this condition is violated, the effect of an isolated occurrence of this kind will generally be only a slight error in the value of the partial derivative given by (2.4). In fact, the zero-boundary condition is known to be a necessary condition for global optimality of a vector quantizer [7].

A gradient descent minimization of $D(Y)$ is given by iteratively updating the codevectors in the direction of the negative gradient. Specifically, at the n-th iteration,

$$\mathbf{y}_i(n+1) = \mathbf{y}_i(n) - \mu \mathbf{g}_i \tag{2.5}$$

for $i = 1, \cdots, N$, where μ, called the step-size, is independent of i, and the gradient is evaluated for the n^{th} codebook $Y(n)$. The optimal value of μ can be found by optimizing the cost function along the direction of $-\mathbf{g}_i$. This is a one-dimensional optimization called a line search. Along this direction the cost as a function of μ is given, using (2.4), by

$$\tilde{D}(\mu) = \tfrac{1}{2} \sum_{i=1}^{N} \sum_{\mathbf{x} \in R_i} \| \mathbf{x} - \mathbf{y}_i + \mu K_i (\mathbf{y}_i - \mathbf{c}_i) \|^2 \quad . \tag{2.6}$$

To find the optimal value of μ, we differentiate $\tilde{D}(\mu)$ with respect to μ and equate the result with zero:

$$\frac{\partial \tilde{D}(\mu)}{\partial \mu} = \sum_{i=1}^{N} \sum_{\mathbf{x} \in R_i} \Big[\mu K_i^2 \| \mathbf{y}_i - \mathbf{c}_i \|^2 + K_i (\mathbf{y}_i - \mathbf{c}_i)^T (\mathbf{x} - \mathbf{y}_i) \Big] = 0. \tag{2.7}$$

The solution to (2.7), denoted by μ_{opt}, is then given by

$$\mu_{opt} = \frac{\sum_{i=1}^{N} K_i^2 \| \mathbf{y}_i - \mathbf{c}_i \|^2}{\sum_{i=1}^{N} K_i^3 \| \mathbf{y}_i - \mathbf{c}_i \|^2} = \frac{\sum_{i=1}^{N} \| \mathbf{g}_i \|^2}{\sum_{i=1}^{N} K_i \| \mathbf{g}_i \|^2} \quad . \tag{2.8}$$

If the centroids $\mathbf{c}_i$ are computed at each update of the algorithm, then *only* one additional Euclidean distance calculation is required for each cell R_i to compute μ_{opt}; namely, the distance between the current value of the codevector $\mathbf{y}_i$ and the centroid $\mathbf{c}_i$.

If we allow a local step-size μ_i for each of the codevectors $\mathbf{y}_i$, then each codevector can be locally adapted towards the centroid using its own step-size. The direction of the adaptation in this case is not the gradient of the cost function defined in (2.1) since the update of the codebook is now given by

$$\mathbf{Y}(n+1) = \mathbf{Y}(n) - \mathbf{U} \nabla_Y D \quad , \tag{2.9}$$

in which $\mathbf{Y}(n) = (\mathbf{y}_1(n), \cdots, \mathbf{y}_N(n))$, $\mathbf{U}$ is the diagonal matrix given by diag($\mu_1, \mu_2, \cdots, \mu_N$), and $\nabla_Y D$ is the gradient of $D(Y)$ in the codebook space. Such an approach of separately tuning the step-size of each component in a gradient descent-like procedure for minimizing a general multi-dimensional function was studied by several researchers in the past (e.g., [3, 4, 5]). However, its advantages in terms of speed of convergence with respect to a true gradient descent (as in (2.5)) are not conclusive. In our case, a solution of (2.7) when μ_i is a function of i can be seen to be

$$\mu_i = \frac{1}{K_i} \quad , \tag{2.10}$$

for which the formula (2.5) for updating the codevectors becomes

$$\mathbf{y}_i(n+1) = \mathbf{y}_i(n) + \frac{1}{K_i} \sum_{\mathbf{x} \in R_i^{(n)}} (\mathbf{x} - \mathbf{y}_i(n)) \quad , \tag{2.11}$$

where $R_i^{(n)}$ is the set of training vectors which are in the nearest-neighbor cell of $\mathbf{y}_i(n)$ at the n^{th} iteration. Eq. (2.11) can be also written as

$$\mathbf{y}_i(n+1) = \frac{1}{K_i} \sum_{\mathbf{x} \in R_i^{(n)}} \mathbf{x} \triangleq \mathbf{c}_i(n) \qquad (2.12)$$

for $i = 1, \cdots, N$, which is exactly the GLA update formula (2.3). Thus, we conclude that the SD-like method of (2.9), in which an optimal local adjustment of the step-size by a line search is individually employed by each of the codevectors, leads to the GLA algorithm, where each codevector is replaced by the new centroid.

It should be noted that for optimization problems in general, tuning each of the variables separately as in (2.9), does not always guarantee a descent. However, for the GLA it can be verified that the cost function of the codebook $Y(n)$ can be written as

$$D(Y(n)) = \tfrac{1}{2} \sum_{i=1}^{N} K_i \, \| \mathbf{y}_i(n) - \mathbf{c}_i(n) \|^2 + \tfrac{1}{2} \sum_{i=1}^{N} \sum_{\mathbf{x} \in R_i^{(n)}} \| \mathbf{x} - \mathbf{c}_i(n) \|^2 \quad , \qquad (2.13)$$

where the second term is $D(Y(n+1))$, by (2.12). Since the first term is always positive (unless $\mathbf{y}_i(n)$ is the centroid of $R_i^{(n)}$) we get $D(Y(n)) > D(Y(n+1))$, which assures descent.

3. Partial Conjugate Gradient Update

We now consider the minimization of $D(Y)$ via the Conjugate Gradient algorithm. In this case, instead of updating the codevectors in a fixed direction, namely, that of the negative gradient, the direction taken in the codebook space will be dynamically updated at each step. We therefore end up with two recursions, one for the codevectors and the other for the directions. In the quadratic case, this algorithm is known to converge in a finite number of steps [1]. In the non-quadratic case, a reinitialization technique is often used so that after every N steps a gradient descent step is performed. The PCG algorithm is generally faster than the steepest descent algorithm since it effectively eliminates the influence of the N largest eigenvalues of the Hessian matrix, which are the major reason for the slow convergence of the steepest descent method. Applying the PCG method to VQ design is achieved by the following recursions. Instead of taking a step in the direction of the gradient as in (2.5), the codevectors are updated in a direction $\mathbf{d}_i(n)$ with a step-size γ_n,

$$\mathbf{y}_i(n+1) = \mathbf{y}_i(n) + \gamma_n \, \mathbf{d}_i(n) \qquad (3.1)$$

for $i = 1, \cdots, N$. The direction $\mathbf{d}_i(n)$ at the n-th iteration is determined by the recursion

$$\mathbf{d}_i(n+1) = -\mathbf{g}_i(n+1) + \zeta_n \, \mathbf{d}_i(n) \qquad (3.2)$$

$n = 0, \cdots, R-1$, which is initialized to $\mathbf{d}_i(0) = -\mathbf{g}_i(0)$, and the gradient $\mathbf{g}_i(n)$ is given according to (2.4). The quantity R is called the *restart rate*, and the algorithm is restarted every R iterations by setting n to zero, initializing $\mathbf{d}_i(0)$ to $-\mathbf{g}_i(0)$, and renaming $\mathbf{y}_i(R)$ to be $\mathbf{y}_i(0)$. In a sweep of R iterations, the first step of the PCG is thus a steepest descent step in the direction of $-\mathbf{g}_i(0)$, and the other $R-1$ steps are conjugate gradient steps. The iteration index n is reinitialized to zero after R steps. The choice of R depends upon the surface of the cost function. When this surface is quadratic (or nearly quadratic) R should be large since the direction update of (3.2) is derived for a quadratic function. When this surface is not even approximately quadratic, R should be small since otherwise the directions computed by the recursion of (3.2) will generally be ineffective. In VQ design, R should be determined by practice, depending on the source, the vector dimension, and the size of the codebook.

In the quadratic case there is an explicit expression for the quantity ζ_n in (3.2). For the non-quadratic case, several propositions for the value of ζ_n are known in the literature. We use the Fletcher-Reeves formula [6] which, adapted to our problem, may be expressed as

$$\zeta_n = \frac{\sum_{i=1}^{N} \| \mathbf{g}_i(n+1) \|^2}{\sum_{i=1}^{N} \| \mathbf{g}_i(n) \|^2} \quad . \qquad (3.3)$$

To optimize the step-size γ_n in (3.1), we follow a similar derivation as used in (2.6)-(2.8). Namely, γ_n is determined as the best choice for the step-size which minimizes the cost function along the direction $\mathbf{d}_i(n)$ taken at time instant n. By differentiating $D(\gamma_n)$ with respect to γ_n as was done is (2.7), the optimal PCG step-size is obtained as

$$\gamma_n = \frac{-\sum_{i=1}^{N} \mathbf{d}_i(n)^T \mathbf{g}_i(n)}{\sum_{i=1}^{N} K_i \, \| \mathbf{d}_i(n) \|^2} \quad . \qquad (3.4)$$

Hence, the additional computation with respect to the conventional GLA is basically *only* three additional inner products for each codevector; the computation of $\mathbf{d}_i(n)^T \mathbf{g}_i(n)$, and the norms of $\mathbf{g}_i(n)$ and $\mathbf{d}_i(n)$. Since the number of distance evaluations required to compute the new centroids in the GLA in each iteration is generally large (K_i distance evaluations for each codevector) the computational requirements of the PCG per iteration are basically the same as those of the GLA. Thus, when comparing the two methods, the number of iterations required to converge may serve as an indicator for the speed of convergence. A block diagram of the PCG algorithm for VQ design is shown in Figure 1.

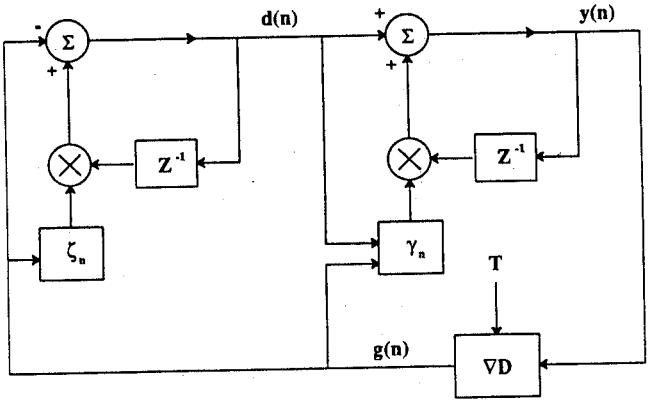

Fig. 1: Block diagram of VQ design procedure using the Partial Conjugate Gradient Algorithm.

We finally give an outline of the PCG algorithm for VQ design:

1. For $n = 0$ perform a gradient descent according to (3.1) with $\mathbf{d}_i(0) = -\mathbf{g}_i(0)$.

2. For $n=1, \cdots, R$, perform $R-1$ conjugate gradient steps as follows:
   2.1. Repartition the training set and compute K_i and the centroids $\mathbf{c}_i(n)$ for $i=1, \cdots, N$.
   2.2. Compute the gradients $\mathbf{g}_i(n)$ according to (2.4) and store their norm.
   2.3. Compute ζ_{n-1} according to (3.3).
   2.4. Compute the new direction according to (3.2).
   2.5. Compute the step size according to (3.4).
   2.6. Compute the new codevectors according to (3.1).
3. Initialize n to zero, rename $\mathbf{y}_i(R)$ to be $\mathbf{y}_i(0)$, and repeat until the cost change in one sweep of R steps is less than some specified threshold.

4. Experimental Results

The partial conjugate gradient algorithm described by equations (3.1) and (3.2) was implemented for various reset values (R) and tested in comparison to the generalized Lloyd algorithm. The reset values chosen were those in the range from 1 to 7, with $R=1$ corresponding to a gradient descent

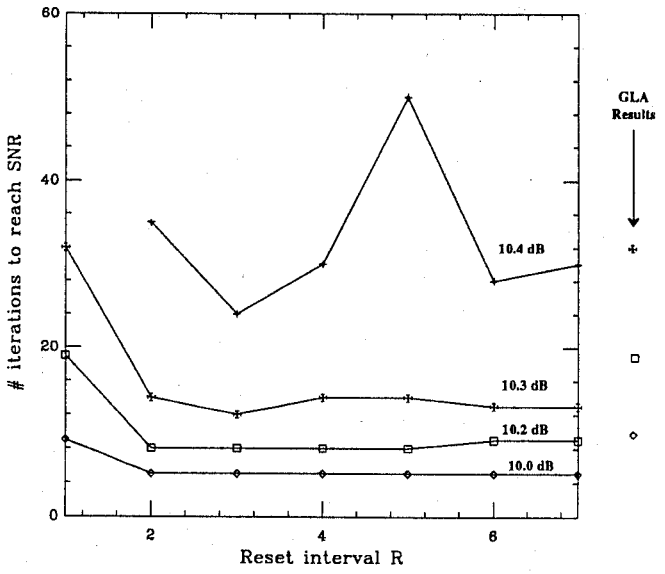

Fig. 2: Plot of number of iterations needed to achieve various SNRs as a function of reset interval (R) for the partial conjugate gradient method of VQ design. Codebook has 16 codevectors of dimension 4, and the source is first-order gauss-Markov of the form $\mathbf{x}_n = .9\mathbf{x}_{n-1} + \mathbf{w}_n$, where $\mathbf{w}_n$ is an i.i.d. Gaussian noise process.

algorithm, and $R=7$ somewhat approximating a true conjugate gradient descent (i.e. $R=\infty$).

The PCG was executed for a variety of different sources, and the results were reasonably consistent. For each instance of the PCG algorithm (i.e. a different value of R), the number of iterations that the algorithm ran before the quantizer's SNR surpassed certain thresholds was recorded. Figure 2 shows typical results of the PCG's performance. The results are based on a a first-order Gauss-Markov source of the form $\mathbf{x}_n = 0.9\mathbf{x}_{n-1} + \mathbf{w}_n$, where $\mathbf{w}_n$ is an i.i.d. Gaussian noise process. The thresholds in this case were chosen in dB as 10.0, 10.2, 10.3, and 10.4. The performance of the GLA for each threshold is shown to the right of the graph. Neither the GLA nor the PCG with $R=1$ were able to achieve the 10.4 dB threshold, and thus the corresponding points on the point were omitted. It can be seen that the PCG generally outperforms the GLA as well as a true gradient descent.

5. Conclusions

A new technique for VQ design is presented based on the Partial Conjugate Gradient algorithm. The complexity per iteration is roughly equivalent to that of the GLA. However, in most cases, either a faster convergence was achieved for the PCG algorithm with respect to the GLA, or a better SNR was obtained for the same number of iterations. The PCG algorithm appears to offer a more efficient vector quantizer design technique than the GLA.

References

[1] D.G. Luenberger, *Linear and nonlinear programming*, Addison-Wesley, Reading, Mass., 1984.

[2] Y. Linde, A. Buzo and R. M. Gray, "An Algorithm for Vector Quantizer Design", *IEEE Trans. Communications*, COM-28, 84-95, January 1980.

[3] A H. Kesten, "Accelerated stochastic approximation", *Annals of Mathematical Statistics*, V0l. 29, 41-59, 1958.

[4] R.A. Jacobs, "Increased rates of convergence through learning rate adaptation", *Journal of Neural Networks*, Vol-1, 295-307, 1988.

[5] G.N. Saridis, "Learning applied to successive approximation algorithms", *IEEE Trans. on Systems Science and Cybernetics*, SSC-6, 97-103, 1970.

[6] R. Fletcher and C.M. Reeves, "Function minimization by conjugate gradients", *Computer J.*, Vol. 7, 149-154, 1964.

[7] A. Gersho and R.M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Boston, 1990, chapter 11.