A New Initialization Technique for VQ Codebook Design

Ioannis Katsavounidis and C.-C. Jay Kuo and Zhen Zhang

Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089-2564

Abstract

The generalized Lloyd algorithm plays an important role in the design of vector quantizers (VQ) and in feature clustering for pattern recognition. In the VQ context, this algorithm provides a procedure to iteratively improve a codebook and results in a local minimum which minimizes the average distortion function. In this research, we propose an efficient method to obtain a good initial codebook that can accelerate the convergence of the generalized Lloyd algorithm, achieve a better local minimum and also result into lower entropy. Thus, it can achieve an overall better performance since the resulting rate-distortion pair is closer to the rate-distortion function of the input source.

1 Introduction

A vector quantizer Q of dimension k and size N is a mapping from a vector in k-dimensional Euclidean space R^k into a finite set C containing N output or reproduction points. We can write it mathematically as $Q : R^k \rightarrow C$ where $C = \{y_1, y_2, \ldots, y_N\}$ and $y_i \in R^k$. The set C is called the codebook and y_i, $1 \leq i \leq N$, are called the code vectors or codewords. To measure the vector quantizer performance, a distortion measure $d(x, Q(x))$ has to be defined in association with any input vector x and its reproduction vector $Q(x)$. With such a measure, one can quantify the performance of a vector quantizer by either the average distortion $D = E[d(x, Q(x))]$ or the worst-case distortion $D_{\max} = \max_x d(x, Q(x))$. To permit tractable analysis and easy evaluation, the distortion measure is often chosen to be the squared error $d(x, Q(x)) = ||x - Q(x)||^2$.

The design of an optimal quantizer is to seek the codebook that minimizes the average distortion over all possible codebooks. It can be easily shown that the optimal quantizer must satisfy the following two conditions. First, it must be a nearest neighbor quantizer, i.e. it assigns to an arbitrary vector the codeword that is closest to it. Second, for a given partition of the feature space, it must satisfy the centroid condition, i.e. each codeword must be the centroid of the vectors that are mapped to it. The above two optimality conditions provide an algorithm for the design of a locally optimal codebook with iterative codebook improvement. This algorithm is known as the generalized Lloyd iteration where each iteration consists of the following two steps.

1. Given a codebook $C_m = \{y_i; i = 1, \ldots, N\}$ obtained from the mth iteration, find the optimal partitioning of the space R^k. That is, use the nearest neighbor condition to form the nearest neighbor cells $R_i = \{x : d(x, y_i) < d(x, y_j); j \neq i\}$.

2. Use the centroid condition to update the codebook $C_{m+1} = \{\text{centroid}(R_i); i = 1, \ldots, N\}$, which is the optimal reproduction codebook based on the cells found in (1).

The above algorithm is an iterative optimization procedure based on the method of alternating projections and, therefore, leads to a local minimum. The generalized Lloyd algorithm is also known as the *k-means algorithm* after MacQueen [3] who studied it as a statistical clustering procedure. It is sometimes referred to as the LBG algorithm in the data compression literature [2].

It has been observed that both the convergence rate of the generalized Lloyd iteration and the performance of the converged codebook depend on the initial codebook C_0. Thus, it is important to find a good initial codebook. Many different initialization methods have been proposed, and a thorough survey of these methods can be found in [1]. After a brief review of existing initialization schemes in Section 2, we will propose a new initialization technique which leads to a fast convergence behavior and a converged codebook with an

1058-6393/95 $4.00

excellent performance in terms of average or worst-case distortion in Section 3. Finally, in Section 4, experimental results will be presented, comparing the new initialization technique with existing methods.

2 Review of Existing Initialization Techniques

We comment on the strength and weakness of several commonly used initialization methods below.

(1) *Random Coding.* Choose randomly N of the training vectors as the initial codewords. Even though it is very simple, it can yield a very bad initial codebook which requires much more generalized Lloyd iterations for convergence. It can also lead to a very "bad" local minimum.

(2) *Pruning.* Start by putting one (usually the first) training vector into the codebook and then keep on adding those vectors that are further than some threshold from all the vectors that are already in the codebook. Stop when the desired size for the codebook has been reached. It has the advantage that the initial codebook consists of vectors that are training vectors, thus providing a guarantee that at no point of the GLA iteration we will get an empty class. Furthermore, those initial codewords, being far apart from each other, are much more likely to belong to different classes. An obvious disadvantage, although, is the choice of the threshold that can result in either a codebook of size smaller than the desired number of code vectors (too large threshold) or a codebook consisting of the first few vectors from the training sequence that are not representative of the entire training sequence (too small threshold). In both cases, the threshold value should be adjusted and the pruning algorithm should be applied again.

(3) *Pairwise Nearest Neighbor Design.* This method starts with the entire training sequence as the codebook and then at each stage groups together those classes that, when grouped, result into the smallest increase of the average distortion. This method provides very good results but is almost forbidden for practical applications, because of the very high computational complexity. Equitz [4] proposed a reduced-complexity pairwise nearest neighbor method that produced better results than random initialization.

(4) *Product Codes.* Start with a scalar quantizer of appropriate resolution on each dimension of the vector space and then take the Cartesian product of these scalar quantizers as the initial codebook for the vector quantizer. It is quite simple, but provides poor results. Furthermore, it is limited to vector quantizers of integer resolution.

(5) *Splitting.* This method produces increasingly larger codebooks from small ones. We start with a resolution 0 (i.e. just one centroid) codebook. This vector is the centroid of the entire training sequence. To obtain an initial codebook for a resolution 1 (i.e. two centroids) codebook, we "split" this initial centroid, say y_0, into two codewords, $y_0 + d$ and $y_0 - d$ where d is the vector whose components are the standard deviations on each dimension. After applying the GLA on this new codebook of size 2, we split again to obtain a codebook of size 4 and so on. This algorithm gives very good results. The only disadvantages are the fact that it is applicable to integer resolution quantizers only and that its complexity is moderate and depends on the number of GLA iterations we take on the intermediate stages. Still, its overall performance has made it the favorite choice among researchers in the field of vector quantization.

To improve the performance of the VQ codebook, methods other than the standard GLA have been examined by researchers. Yair *et al.* [5] suggested an on line (or serial) version of the GLA which achieved better results than some implementations of the GLA. Zeger *et al.* [6] proposed to use stochastic relaxation, such as the simulated annealing method, to achieve a better result than GLA. These methods show an average improvement of 0.3dB in codebook performance, but they require significantly more execution time than the GLA.

3 New Initialization Technique

The idea behind our technique is similar to that of pruning. That is, we target those vectors in the training sequence that are the most distant from each other, because they are more likely to belong to different classes. Let v_i, $i = 1, \ldots, M$ be the training sequence of vectors. We define as the distance of a vector x to a codebook C and denote it by $dist(x, C)$ to be the minimum of all the distances between x and every codeword y_j, $1 \leq j \leq N$, i.e.

$$dist(x, C) := min\{d(x, y_j),\ 1 \leq j \leq N\}$$

To better explain the initialization technique and also give its optimal (in terms of computational complexity) implementation, we will utilize two arrays. One is called the *distance array*, $Distance(i)$, $i = 1, \ldots, M$. This array is used to store the minimum distance of

the vector v_i from the codebook, as defined above, i.e.

$$Distance(i) = dist(v_i, C) = min\{d(v_i, y_j), 1 \leq j \leq N\}.$$

Associated with the distance array is the *index array*, $Index(i)$, $i = 1, \ldots, M$. The content of this array is the index of the codeword from the codebook that is closest to the vector v_i, i.e.

$$Index(i) = argmin\{d(v_i, y_j),\ 1 \leq j \leq N\}.$$

So, at every point, for each vector v_i we know its nearest neighbor codeword, which is $y_{Index(i)}$, and also the corresponding minimum distance $d(v_i, y_{Index(i)}) = Distance(i)$. Our procedure can now be stated as follows.

1. *Initialization (I).* Calculate the norms of all the vectors in the training sequence. Choose the vector with the maximum norm as the first codeword in the initial codebook, i.e.

   $$y_1 := \{v_j :\ ||v_j|| \geq ||v_k||,\ \forall k \in \{1, \ldots, M\}\}.$$

   Set the size of the codebook, i, equal to one and so $C = \{y_1\}$.

2. *Initialization (II).* Calculate the distances of all the vectors in the training sequence from the first codeword and store them in the distance array, i.e.

   $$Distance(j) := d(v_j, y_1), \quad j = 1, \ldots, M.$$

   At the same time, set all the entries of the index array equal to one, i.e.

   $$Index(j) := 1, \quad j = 1, \ldots, M.$$

3. *Iteration (I).* Find the maximum entry in the distance array and announce the corresponding training vector as the $(i+1)$th codeword, i.e.

   $$y_{i+1} := \{v_j :\ Distance(j) \geq Distance(k),$$
   $$\forall k \in \{1, \ldots, M\}\}.$$

4. *Iteration (II)* Loop over all vectors in the training sequence. For each one of the training vectors, calculate its distance from the new -$(i+1)$th-codeword. If it is less than the entry in the distance array then update both the distance and the index array correspondingly, i.e.

   for $j = 1, \ldots, M$
   $t := d(v(j), y_{i+1})$
   if $(t < Distance(j))$ then
   $\{Distance(j) = t; \quad Index(j) = i + 1\}$.

5. *Termination condition.* Set $i := i + 1$; if $i = N$, stop, else goto step three.

The essence of the above procedure can be briefly explained below. At the first step we choose the vector that has the maximum distance from the origin. At the second step we choose the vector that has the maximum distance from the first codeword. At the third step we choose the vector that has the maximum distance from both the first and the second codewords. At the ith step we choose the vector the has the maximum distance from the codebook consisting of the first $(i-1)$ vectors. This is the reason one can call this new initialization technique as the *maximum distance initialization*. This can also be seen as using the vector that is the most different from existing code vectors as the new codeword.

The proposed initialization scheme has several attractive properties. First, it can be applied to arbitrary codebook sizes (not only integer resolutions, as is the case for product codes and splitting). Second, there is no need for the specification of a threshold, like the pruning algorithm. Third, since it takes exactly N iterations to obtain a codebook of size N and each iteration requires M distance calculations, consequently th entire initialization procedure has a complexity of $O(MN)$. This is exactly the same as the complexity of one generalized Lloyd iteration. Since, on top of that, during the initialization phase we keep record of the index, $Index(i)$ for each training vector together with its the minimum distance, after the end of the initialization phase we also obtain the initial partition. This means that by doing the proposed initialization algorithm, we are in fact performing the first generalized Lloyd iteration.

4 Performance Evaluation

We find it difficult to provide a performance analysis of the proposed new method, and will instead present some experimental results to illustrate its performance. We applied the new method to a set of training sequences obtained from 3 test images. They are the well known baboon, Lena and boat images from the USC image database which have a size of 512×512 pixels and 256 gray scales per pixel. We examined different block sizes including 2×2, 4×4 and 8×8.

Our method was compared with the splitting method [1] which is in fact the primary competitor. We restricted our experiments to integer resolution i.e.

codebooks of size $N = 2^m$ for some integer m. However, it must be emphasized that this is a restriction of the splitting and *not* of our method. We allowed 2 generalized Lloyd iterations in the intermediate stages of the splitting process. Since the complexity of one generalized Lloyd iteration at resolution 2^i is $2^i M$, the complexity of the whole initialization process is $2\sum_{k=0}^{m-1} 2^k M = 2M(2^m - 1)$, which is equivalent to the computational complexity of 2 generalized Lloyd iterations at the finest (i.e. 2^m) level.

We report the results for the baboon image in Fig. 1 and Table 1. In Fig. 1, we plot (in semi-logarithmic scale) the MSE value versus the number of generalized Lloyd iteration with different block sizes by choosing $N = 512$. Recall that the initialization with the splitting method is equivalent to 2 generalized Lloyd iterations while our method does not add extra cost. Thus, to make a fair comparison, the solid curve (our method) starts with 0 iteration while the dashed curve (splitting) starts with 2 iterations. We see that the MSE decays very quickly (in just 1 iteration) for our method and has a lower converged value. We also list the MSE values for various codebook and block sizes after 20 generalized Lloyd iterations in Table 1. We see that our method provides a better codebook with lower MSE values for all cases. To characterize the gain, we computed the ratios of MSE values of two methods and expressed them in dB. When the vector dimension k or the codebook size N becomes larger, the gain is more pronounced. Generally speaking, the improvement is very impressive. The same results were also observed for the Lenna and boat test images, as this can be seen on Tables 2 and 3 respectively.

Besides the MSE, there is another very important criterion for the evaluation of a vector quantizer, namely, the entropy rate achieved by it, usually expressed in bits. To calculate the entropy, we first need to define the empirical distribution of the training sequence with respect to the (final) codebook:

$$p_i = \frac{\text{no. of vectors classified to codeword } y_i}{\text{total no. of input vectors}}.$$

Then, the entropy of the encoded image (in bits per pixel) based on the empirical distribution of the code vectors is given by

$$H = -\frac{1}{k}\sum_{i=1}^{N} p_i log_2(p_i)$$

where N is the codebook size and k is the vector dimension. We list the entropy values for various codebook and block sizes after 20 generalized Lloyd iterations in Tables 4-6. We see that our method provides a better codebook with lower entropy values for all cases; the average improvement being 12%. Depending on the specific VQ application, this may turn out to be even more important than the reduction in the MSE. The improved values for both MSE and entropy figures indicate that the new proposed algorithm results in a codebook that is much better *matched* to the training sequence than that achieved by existing methods, and thus can be used for the vector quantization of sources with similar statistics, providing far superior results.

Table 1: MSE values of codebooks obtained in 20 generalized Lloyd iterations for the baboon image.

	4 × 4 blocks			8 × 8 blocks		
N	Split.	New	Gain	Split.	New	Gain
128	375.8	361.3	0.17	641.9	614.7	0.19
256	329.6	311.4	0.25	581.8	533.8	0.37
512	283.0	265.0	0.29	492.9	430.5	0.59
1024	238.2	217.6	0.39	347.9	278.7	0.96
2048	188.6	163.1	0.63	135.6	96.7	1.40

Table 2: MSE values of codebooks obtained in 20 generalized Lloyd iterations for the boat image.

	4 × 4 blocks			8 × 8 blocks		
N	Split.	New	Gain	Split.	New	Gain
128	99.4	99.7	−0.01	204.6	196.4	0.18
256	81.3	78.6	0.15	161.6	149.4	0.34
512	64.1	61.1	0.21	112.7	96.1	0.69
1024	48.6	44.2	0.42	57.5	47.7	0.81
2048	32.7	27.9	0.69	16.9	14.6	0.62

5 Conclusion

An efficient initialization technique for the generalized Lloyd iteration was proposed in this work. It reduces the computational complexity, achieves a better local minimum and also produces codebooks of lower entropy. Thus, it provides an excellent choice for the implementation of the generalized Lloyd algorithm in both vector quantization and unsupervised clustering applications.

Table 3: MSE values of codebooks obtained in 20 generalized Lloyd iterations for the Lenna image.

	4 × 4 blocks			8 × 8 blocks		
N	Split.	New	Gain	Split.	New	Gain
128	66.5	67.7	−0.08	138.2	128.8	0.31
256	52.6	52.5	0.01	105.4	92.8	0.55
512	41.4	39.7	0.18	69.8	56.7	0.90
1024	30.9	27.6	0.49	31.1	24.9	0.96
2048	20.4	17.0	0.81	7.4	6.7	0.46

Table 4: Entropy values (in bits per vector) of codebooks obtained in 20 generalized Lloyd iterations for the baboon image.

	4 × 4 blocks			8 × 8 blocks		
N	Split.	New	Diff.	Split.	New	Diff.
128	6.61	6.35	4.1%	6.30	6.00	5.0%
256	7.51	7.13	5.3%	7.04	6.74	4.5%
512	8.45	7.86	7.5%	7.85	7.45	5.4%
1024	9.37	8.56	9.5%	8.79	8.30	5.9%
2048	10.31	9.36	10.1%	9.99	9.72	2.8%

Table 5: Entropy values (in bits per vector) of codebooks obtained in 20 generalized Lloyd iterations for the boat image.

	4 × 4 blocks			8 × 8 blocks		
N	Split.	New	Diff.	Split.	New	Diff.
128	6.16	5.28	16.7%	6.01	5.02	19.7%
256	7.12	6.03	18.1%	6.78	5.72	18.5%
512	8.12	6.68	21.6%	7.68	6.54	17.4%
1024	9.11	7.51	21.3%	8.65	7.89	9.6%
2048	10.08	8.65	16.5%	9.92	9.77	1.5%

Table 6: Entropy values (in bits per vector) of codebooks obtained in 20 generalized Lloyd iterations for the Lenna image.

	4 × 4 blocks			8 × 8 blocks		
N	Split.	New	Diff.	Split.	New	Diff.
128	6.38	5.28	20.8%	6.34	5.24	21.0%
256	7.31	5.95	22.9%	7.18	5.85	22.7%
512	8.29	6.65	24.7%	7.95	6.73	18.1%
1024	9.24	7.47	23.7%	8.81	7.92	11.2%
2048	10.19	8.57	18.9%	10.04	9.77	2.8%

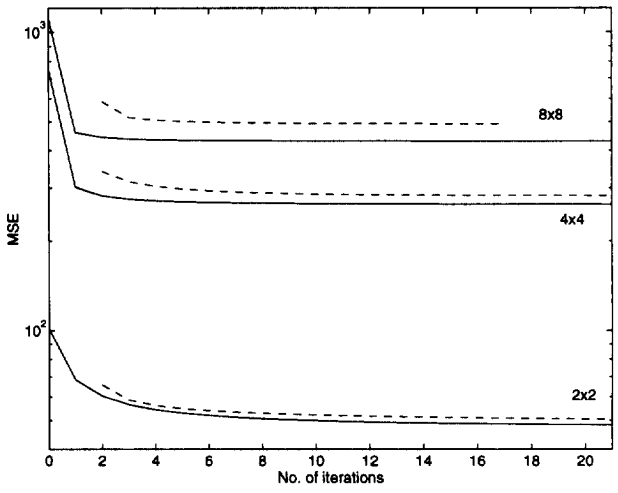

Figure 1: Convergence history of generalized Lloyd iteration with the splitting (dashed) and proposed new (solid) initialization methods.

References

[1] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, 1992.

[2] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Comm.*, Vol. COM-28, pp. 84–95, 1980.

[3] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the Fifth Berkeley Symposium on Math. Stat. and Prob.*, vol. 1, pp. 281–296, 1967.

[4] W. H. Equitz, "A New Vector Quantization Clustering Algorithm," *IEEE Trans. on ASSP*, Vol. 37, No. 10, Oct. 1989, pp. 1568–1575.

[5] E. Yair, K. Zeger, and A. Gersho, "Competitive Learning and Soft Competition for Vector Quantizer Design," *IEEE Trans. on ASSP*, Vol. 40, No. 2, Feb. 1992, pp. 294–309.

[6] K. Zeger, J. Vaisey, and A. Gersho, "Globally Optimal Vector Quantizer Design by Stochastic Relaxation," *IEEE Trans. on ASSP*, Vol. 40, No. 2, Feb. 1992, pp. 310–322.