Available online at www.sciencedirect.com
SCIENCE DIRECT
Neural Networks 19 (2006) 694–704
Neural Networks
www.elsevier.com/locate/neunet

An adaptive incremental LBG for vector quantization

F. Shen[a,*], O. Hasegawa[b,c]

[a]Department of Computational Intelligence and System Science, Tokyo Institute of Technology, R2, 4259 Nagatsuta, Midori-ku, Yokohama, 226-8503, Japan
[b]Imaging Science and Engineering Lab., Tokyo Institute of Technology
[c]PRESTO, Japan Science and Technology Agency (JST)

Received 22 October 2004; accepted 17 May 2005

Abstract

This study presents a new vector quantization method that generates codewords incrementally. New codewords are inserted in regions of the input vector space where the distortion error is highest until the desired number of codewords (or a distortion error threshold) is achieved. Adoption of the adaptive distance function greatly increases the proposed method's performance. During the incremental process, a removal–insertion technique is used to fine-tune the codebook to make the proposed method independent of initial conditions. The proposed method works better than some recently published efficient algorithms such as Enhanced LBG (Patane, & Russo, 2001) for traditional tasks: with fixed number of codewords, to find a suitable codebook to minimize distortion error. The proposed method can also be used for new tasks that are insoluble using traditional methods: with fixed distortion error, to minimize the number of codewords and find a suitable codebook. Experiments for some image compression problems indicate that the proposed method works well.

Keywords: LBG; Local quantization error (LQE); Incremental; Removal–insertion; Adaptive distance function

1. Introduction

The purpose of vector quantization (VQ) (Vladimir & Filip, 1997) is to encode data vectors in order to transmit them over a digital communications channel. Vector Quantization is appropriate for applications in which data must be transmitted (or stored) with high bandwidth, but tolerating some loss in fidelity. Applications in this class are often found in speech and image processing such as audio (Paliwal & Atal, 1993), video (Cosman, Gray, & Vetterli, 1996), data compression, pattern recognition (Chan, Nasrabadi, & Mirelli 1996), computer vision (Jolion, Meer, & Bataouche, 1991), medical image recognition (Perlmutter, Perlmutter, Gray, Olshen, & Oehler, 1996), and others.

To create a vector quantization system, we must design both an encoder (quantizer) and a decoder. First, we partition the input space of the vectors into a number of disjoint regions. Then, we must find a prototype vector (a codeword) for each region. When given an input vector, the encoder produces the index of the region where the input vector lies. This index is designated as a channel symbol. The channel symbol is the result of an encoding process and is transmitted over a binary channel. At the decoder, the channel symbol is mapped to its corresponding prototype vector (codeword). The transmission rate is dependent on the number of quantization regions. Given the number of regions, the task of designing a vector quantizer system is to determine the regions and codewords that minimize the distortion error.

Many vector quantization algorithms have been proposed; new algorithms continue to appear. These methods are classifiable into two groups (Patane & Russo, 2001): *k*-means- based algorithms and competitive -learning-based algorithms. Typically, *k*-means-based algorithms are designed to minimize distortion error by selecting a suitable codebook. An exemplary method of this group is the LBG algorithm (Linde, Buzo, & Gray, 1980). Competitive learning-based methods mean that codewords are obtained as a consequence of a process of mutual competition. Typical methods of this kind are self-organizing map

* Tel.: +81 45 924 5180; fax: +81 45 924 5175.
E-mail address: furaoshen@isl.titech.ac.jp (F. Shen).

0893-6080/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2005.05.001

(SOM) (Kohonen, 1982), neural gas (NG) (Martinetz, Berkovich, & Schulten, 1993), and growing neural gas (GNG) (Fritzke, 1995). In this paper, the proposed method is based on the *k*-means-based method; we also adopt some techniques of competitive learning to enhance the performance of the proposed method.

Linde et al. (1980) proposed a popular vector quantization method that is intended to minimize distortion error. The algorithm is known as LBG from the initials of its authors. However, the LBG algorithm is a local search procedure. It suffers from the serious drawback that its performance depends heavily on the initial starting conditions. Many studies have been undertaken to solve this problem. For example, Likas, Vlassis, and Verbeek (2003) proposed a global *k*-means algorithm that is an incremental approach to clustering that dynamically adds one cluster center at a time through a deterministic global search procedure consisting of N (where N represents the data set size) executions of the *k*-means algorithm from suitable initial positions. Compared with traditional *k*-means, this algorithm obtains equal or better results, but suffers from a very high computation load. For that reason, this method is unsuitable for real world data that usually constitute a large data set and consequently require numerous codewords. Both LBG-U (Fritzke, 1997) and Enhanced LBG (ELBG) (Patane & Russo, 2001) algorithms use nearly the same technique to overcome the main drawbacks of clustering algorithms: the dependence of initial starting conditions. Both methods define one parameter—the utility of codeword—and move low-utility codewords to positions near high-utility codewords. Results show that ELBG performs better than LBG-U in quantization quality and computation load. In this article, LBG and ELBG are used as benchmark algorithms for comparison with the proposed method.

In general, algorithms of vector quantization focus on solving this kind of problem: given the number of codewords, determine the quantization regions and the codewords that minimize the distortion error. In some applications, for example, to set up an image database with the same distortion error for every image in the database, the quantization problem becomes this kind of problem: given the distortion error, minimize the number of codewords and determine the quantization regions and codewords. As far as we know, no *k*-means-based method can accomplish this task.

This paper presents a new vector quantization algorithm called adaptive incremental LBG. It can accomplish both tasks: predefining the number of codewords to minimize distortion error; and predefining the distortion error threshold to minimize the number of codewords. We improved LBG in the following aspects. (1) By introducing some competitive mechanism, the proposed method incrementally inserts a new codeword near the codeword that contributes most to error minimization. This technique renders the proposed method as suitable for the second task: the transmission rate (or compression ratio) can be controlled using the distortion error. (2) By adopting an adaptive distance function to measure the distance between vectors, the proposed method can obtain superior results to the use of Euclidean distance. (3) By periodically removing codewords with no contribution or the lowest contribution to error minimization, the proposed method fine-tunes the codebook and makes the proposed algorithm independent of initial starting conditions.

Taking image compression as a real-world example, we conduct some experiments to test the proposed method. Experimental results show that the proposed method is independent of initial conditions. It is able to find a better codebook than previous algorithms such as ELBG. Furthermore, it is capable of finding a suitable number of codewords with a predefined distortion error threshold.

The paper is organized as follows. Section 2 introduces the basic VQ concepts and LBG algorithm; it presents an analysis of LBG. Section 3 describes improvements to LBG in several aspects and presents experiments to support our improvements, finally describing the proposed method, and Section 4 explains numerous experiments to test the proposed method and compare it with LBG and ELBG.

2. Vector quantization

This section provides some basic concepts of vector quantization, introduces the traditional LBG algorithm, and presents an analysis of the LBG algorithm.

2.1. Definition

A vector quantizer Q is a mapping of a l-dimensional vector set $X=\{x_1,x_2,\ldots,x_n\}$ into a finite l-dimensional vector set $C=\{c_1,c_2,\ldots,c_m\}$, where $l\geq 2$ and $m\ll n$. Thus

$$Q : X \rightarrow C \tag{1}$$

C is called a codebook. Its elements $c_1,c_2,\ldots,c_m$ are called codewords. Associated with m codewords, there is a partition $R_1,R_2,\ldots,R_m$ for X, where

$$R_j = Q^{-1}(c_j) = \{x \in X : Q(x) = c_j\}. \tag{2}$$

From this definition, the regions defining the partition are non-overlapping (disjoint) and their union is X. A quantizer is uniquely definable by jointly specifying the output set C and the corresponding partition R_j. This definition combines the encoding and decoding steps as one operation called quantization.

Vector quantizer design consists of choosing a distance function $d(x,c)$ that measures the distance between two vectors x and c. A commonly used distance function is the

squared Euclidean distance

$$d(x,c)=\sum_{i=1}^{l}(x_i-c_i)^2 \tag{3}$$

A vector quantizer is optimal if, for a given value of m, it minimizes the distortion error. Generally, mean quantization error (MQE) is used as the measure of distortion error

$$\mathrm{MQE}\equiv\frac{1}{n}\sum_{i=1}^{m}E_i \tag{4}$$

where

$$E_i=\sum_{j:x_j\in R_i}d(x_j,c_i) \tag{5}$$

is the Local Quantization Error (LQE) of codeword c_i.

Two necessary conditions exist for an optimal vector quantizer—the Lloyd–Max conditions (Lloyd, 1957; Max, 1960).

(1) The codewords c_j must be given by the centroid of R_j:

$$c_j=\frac{1}{N_j}\sum_{i=1}^{N_j}x_i,\quad x_i\in R_j, \tag{6}$$

where N_j is the total number of vectors belonging to R_j.

(2) The partition R_j, $j=1,\ldots,m$ must satisfy

$$R_j\supset\{x\in X: d(x,c_j)<d(x,c_k)\,\forall\, k\neq j\}. \tag{7}$$

This partition is a Voronoi partition; R_j is the Voronoi region of codeword c_j, $j=1,\ldots,m$.

Note that the above two necessary conditions are generalizable for any distance function. In that case, the output points are determined by the generalized centroid, which is the center of mass as determined using a special distance measure function. The Voronoi partition is also determined using that special distance measure function.

2.2. LBG algorithm

An algorithm for a scalar quantizer was proposed by Lloyd (1957). Later, Linde et al. (1980) generalized it for vector quantization. This algorithm is known as LBG or generalized Lloyd algorithm (GLA). It applies the two necessary conditions to inputting data in order to determine optimal vector quantizers.

Given inputting vector data x_i, $i=1,\ldots,n$, distance function d, and initial codewords $c_j(0)$, $j=1,\ldots,m$, the LBG iteratively applies two conditions to produce a codebook with the following algorithm:

Algorithm 2.1. LBG algorithm

(1) Partition the inputting vector data x_i, $i=1,\ldots,n$, into the channel symbols using the minimum distance rule. This partitioning is stored in an $n\times m$ indicator matrix S whose elements are defined as the following

$$s_{ij}=\begin{cases}1, & \text{if } d(x_i,c_j(k))=\min_{\mathrm{p}}d(x_i,c_{\mathrm{p}}(k))\\ 0, & \text{otherwise}\end{cases} \tag{8}$$

(2) Determine the centroids of the Voronoi regions by channel symbol. Replace the old codewords with these centroids:

$$c_j(k+1)=\frac{\sum_{i=1}^{n}s_{ij}x_i}{\sum_{i=1}^{n}s_{ij}},\quad j=1,\ldots,m \tag{9}$$

(3) Repeat steps 1 and 2 until no c_j, $j=1,\ldots,m$ changes anymore.

Note that the two conditions (Eqs. (6) and (7)) only give necessary conditions for an optimal VQ system. Consequently, the LBG solution is only locally optimal and might not be globally optimal. The quality of this solution depends on the choice of initial codebook.

2.3. Considerations about LBG algorithm

The LBG algorithm requires initial values for codewords c_j, $j=1,\ldots,m$. The quality of the solution depends on this initialization. Obviously, if the initial values are near an acceptable solution, a higher probability exists that the algorithm will find an acceptable solution. However, poorly chosen initial conditions for codewords lead to locally optimal solutions.

Patane and Russo (2001) provided a detailed analysis of poorly chosen initial conditions. Fig. 1 shows one example of badly positioned codewords. If an initial codeword is generated as an empty cell (the 2nd codeword in Fig. 1), because all the elements of the data set are nearer to other codewords, following the iterations of LBG, the 2nd codeword cannot move and will never represent any

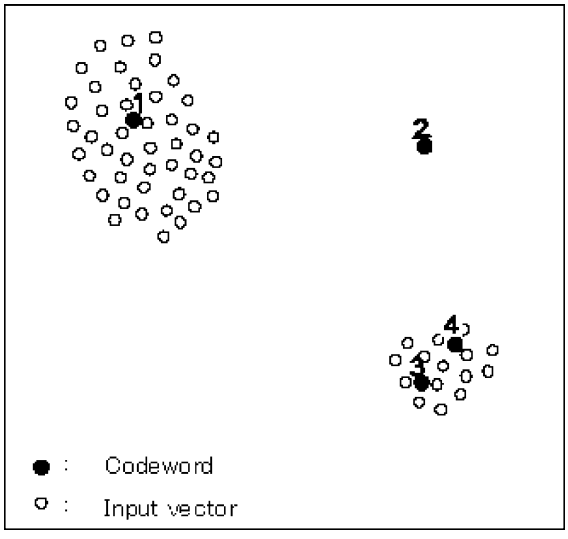

Fig. 1. Poorly initialized codewords.

elements. We say that the 2nd codeword is useless because it has no contribution for the reduction of distortion error. Another problem is that too many codewords are generated for small clusters, but few codewords are generated for large clusters in the initial stage. In Fig. 1, the smaller cluster has two codewords (the 3rd codeword and the 4th codeword), in the larger cluster, only one codeword (the 1st codeword) exists. Even the elements in the smaller cluster are well approximated by the related codewords, but many elements in the larger one are badly approximated. The 3rd and 4th codewords are said to have a small contribution to the reduction of distortion error; the 1st codeword has a large contribution to the reduction of distortion error. From now on, we use the LQE (defined by Eq. (5)) to measure the contribution for the reduction of distortion error.

In signal processing, the dependence on initial conditions is usually cured by applying the LBG algorithm repeatedly, starting with different initial conditions, then choosing the best solution. Both LBG-U (Fritzke, 1997) and ELBG (Patane & Russo, 2001) define different utility parameters to realize a similar target; they try to identify codewords that do not contribute much to the reduction of distortion error and move them to somewhere near codewords that contribute more to the reduction of distortion error.

Section 3 presents the so-called adaptive incremental LBG algorithm. It is independent of initial conditions. It can function similarly to a traditional LBG, i.e. with a fixed number of codewords, to find a codebook that minimizes the distortion error, and works better than ELBG. It can also work for new tasks: with a fixed distortion error limitation, minimize the number of codewords and generate a suitable codebook.

3. Adaptive incremental LBG

As discussed in Section 2, the targets of the proposed method are:

- To solve the problem caused by poorly chosen initial conditions, as shown in Fig. 1.
- With a fixed number of codewords, find a suitable codebook to minimize distortion error. It must work better than some recently published efficient algorithms such as ELBG.
- With fixed distortion error, minimize the number of codewords and find a suitable codebook.

To realize such targets, we do some improvements for LBG. From now on, we will do some experiments to test such improvements and compare such improvements with LBG algorithm. We take the well-known Lena image ($512\times512\times8$) (Fig. 5) as the test image. The image is divided into 4×4 blocks and the resulting 16,384 16-dimensional vectors are the input vector data. In image compression applications, the peak signal to noise ratio (PSNR) is used to evaluate the reconstructed images after encoding and decoding. The PSNR is defined as

$$\mathrm{PSNR} = 10\log_{10}\frac{255^2}{\frac{1}{N}\sum_{i=1}^{N}(f(i)-g(i))^2}, \tag{10}$$

where f and g, respectively, represent the original image and the reconstructed one. All gray levels are represented with an integer value of [0,255]. The total number of pixels in image f is indicated by N. From our definition, higher PSNR implies lower distortion error; therefore, the reconstructed image is better.

3.1. Improvement I: incrementally inserting codewords

To solve the k-clustering problem, Likas et al. (2003) give the following assumption: an optimal clustering solution with k clusters is obtainable by starting from an initial state with

- the $k-1$ centers placed at the optimal positions for the $(k-1)$-clustering problem and
- the remaining kth center placed at an appropriate position to be discovered.

To find the appropriate position of the kth center, they fully search the input data set by performing N executions of the k-means algorithm (here N means the total number of vectors of the input vector set). In all experiments (and for all values of k) they performed, the solution obtained by the assumption was at least as good as that obtained using numerous random restarts of the k-means algorithm. Nevertheless, the algorithm is a rather computational heavy method and is very difficult to use for a large data set.

We hope that the k-codeword problem can be solved from the solution of $(k-1)$-codeword problem once the additional codeword is placed at an appropriate position within the data set. With this idea, we give Algorithm 3.1 to insert codewords incrementally.

Algorithm 3.1. Incremental LBG: predefining number of codeword

(1) Initialize the codebook C to contain one codeword c_1, where c_1 is chosen randomly from the original data vector set. Predefine the total number of codewords m. Adopt the squared Euclidean distance (defined by Eq. (3)) as the distance measure function.
(2) With codebook C, execute the LBG algorithm (Algorithm 2.1) to optimize codebook C. Record the current number of codewords q, every LQE E_i, and Voronoi region R_i of c_i $(i=1,\ldots,q)$.
(3) If current codewords q are fewer than m, insert a new codeword c_{new} to codebook C, i.e. if $q<m$, do the following to insert a new codeword:
   - Find the codeword whose LQE is largest (*winner*):

$$C_{\mathrm{winner}} = \underset{c_i\in C}{\operatorname{argmax}}\, E_i \tag{11}$$

- Randomly choose one vector c_{new} from the Voronoi region R_{winner} of c_{winner}.
- Add the c_{new} to codebook C

$$C = C \cup c_{new} \tag{12}$$

(4) Go to step 2 to execute LBG with the new codebook. Repeat this process until the required number of codewords m is reached.

In Algorithm 3.1, we do not fully search the original data set to find an appropriate new codeword, but randomly choose a vector from the Voronoi region of *winner*. Even this choice cannot assure that the new codeword position is optimal, but this choice has a higher probability of being optimal than randomly choosing one vector from the original data set. This technique obviates a great computation load and renders this method as amenable to large data sets and real world tasks. We must note that such a choice leads to the results depending on the initial position of new codeword. We will solve this problem in Section 3.3.

We tested Algorithm 3.1 with Lena (512×512×8) and compared the results with LBG in Fig. 2. We performed five runs for Algorithm 3.1 and LBG (Algorithm 2.1), then took the mean of the results as the last result. These results show that, with the same number of codewords, Algorithm 3.1 obtains higher PSNR than LBG. We infer that given the same compression ratio, Algorithm 3.1 achieves better reconstruction quality than LBG.

An additional advantage of Improvement I (incrementally inserting codewords) is that, in order to solve the m-codeword problem, all intermediate k-codeword problems are also solved for $k=1,\ldots,m$. This fact might prove useful in many applications where the k-codeword problem is solved for several values of k.

One important performance advantage of the Improvement I is that we can minimize the number of codewords with the given distortion error limitation. This technique is very useful when we need to encode different data sets with the same distortion error. We simply change Algorithm 3.1 to Algorithm 3.2 to solve such a problem.

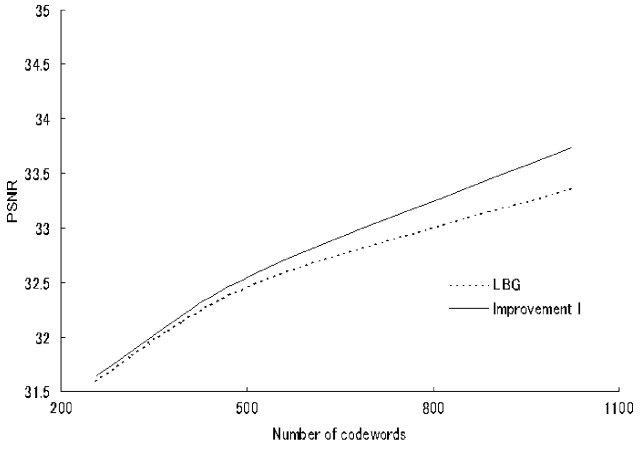

Fig. 2. Comparison results: Improvement I and LBG (Improvement I means incrementally inserting codewords).

Algorithm 3.2. Incremental LBG: predefining the distortion error threshold

(1) Initialize the codebook C to contain one codeword c_1, where c_1 is chosen randomly from an original data vector set. Predefine the distortion error threshold η. Adopt the squared Euclidean distance as the distance measure function.
(2) With codebook C, execute the LBG algorithm (Algorithm 2.1) to optimize codebook C. Record the current MQE ξ, every LQE E_i, and Voronoi region R_i of c_i $(i=1,\ldots,q)$.
(3) If the current MQE ξ is greater than η, insert a new codeword c_{new} to codebook C, i.e. if $\xi>\eta$, do the following to insert a new codeword:
   - Find the codeword whose LQE is largest (*winner*):

$$C_{winner} = \underset{c_i \in C}{\operatorname{argmax}}\, E_i, \tag{13}$$

   - Randomly choose one vector c_{new} from the Voronoi region R_{winner} of c_{winner}.
   - Add the c_{new} to codebook C

$$C = C \cup c_{new}. \tag{14}$$

(4) Go to step 2 to do LBG with the new codebook. Repeat this process until the required distortion error threshold is reached.

3.2. Improvement II: distance measuring function

We expect that the distance between samples and the center of the Voronoi region is *considerably* less than the distance between centers of Voronoi regions, i.e. the within-cluster distance is *considerably* less than the between-cluster distance. Generally, the squared Euclidean distance (defined by Eq. (3)) is used as a measure of distance. This particular choice is justified if the feature space is isotropic and the data are spread roughly evenly along all directions. For some particular applications (i.e. speech and image processing), more specialized distance functions exist (Gray, 1984).

In Algorithms 3.1 and 3.2, if we use the definite measure of distance, such as the squared Euclidean distance, following the increasing of number of codewords, the distance between codewords, i.e. the distance between the center of Voronoi regions (between-cluster distance) shrinks. Even if the within-cluster distance (distance between samples and codeword) is less than the between-cluster distance, it is difficult for the within-cluster distance to be *considerably* less than the between-cluster distance. To solve this problem, we adaptively measure the distance between vectors. Following the increasing of codewords, the distance between vectors will also be increased.

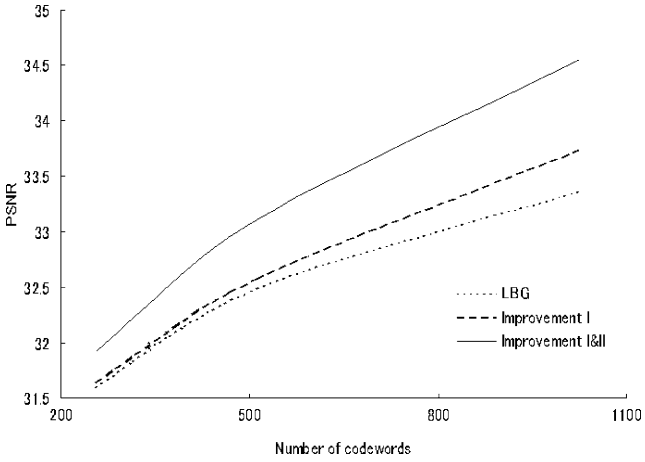

Fig. 3. Comparison results: LBG, Improvement I, and Improvements I and II (Improvements I and II include incrementally inserting codewords, and adopting the adaptive distance function).

As an example, we define an adaptive distance function for image compression tasks: assume that the current number of codewords is q, the distance function $d(x,c)$ is defined as

$$p = \log_{10} q + 1 \tag{15}$$

$$d(x,c) = \left(\sum_{i=1}^{l} (x_i - c_i)^2 \right)^p. \tag{16}$$

In this definition, following the increase in the number of codewords q, p is increased. Consequently, the measure of distance $d(x,c)$ is also increased. This technique ensures that the within-cluster distance is *considerably* less than the between-cluster distance.

In Algorithm 3.1, we use the adaptive distance function to take the place of the squared Euclidean distance, compare the adaptive distance method with the squared Euclidean distance method and LBG. Lena (512×512×8) is tested, and we take the mean of five runs for comparison. Fig. 3 shows the results of comparison: the adaptive distance function works much better than the squared Euclidean distance function. We must note that for a different data set, a different distance function might be needed. Herein, we give just one example of a distance function for image compression tasks.

3.3. Improvement III: removing and inserting codewords

The main shortcoming of LBG is its dependence on initial conditions. In Algorithms 3.1 and 3.2, we randomly choose a vector from the Voronoi region of the *winner* as the new codeword. This choice cannot assure that it is the optimal choice. The results depend on the initial position of the new codeword. Here, we use a removal–insertion process to fine-tune the codebook generated by Algorithm 3.1 or Algorithm 3.2 (with adaptive distance function) to render the algorithms as independent of initial conditions.

Gersho (1986) provided some theorems to show that with high resolution, each cell contributes equally to the total distortion error in optimal vector quantization. Here, high resolution means that the number of codewords tends to be infinite. Chinrungrueng and Sequin (1995) proved experimentally that this conclusion maintains certain validity even when the number of codewords is finite. Based on this conclusion, ELBG (Patane & Russo, 2001) defines the 'utility index' for every element to help fine-tune the codebook and achieve better results than some previous works.

Here we do not define any utility parameter; we only use the LQE as the benchmark of removal and insertion. This removal–insertion is based on this assumption: the LQEs of every codeword will be mutually equal. According to this assumption, we delete the codewords with 0 LQE (we say it has no contribution for the decreasing of distortion error) or the codeword with lowest LQE (*loser*). Otherwise, we insert a vector in the *winner*'s Voronoi region as the new codeword and repeat this process until the termination condition is satisfied.

We must determine how to remove and insert codewords, and what the termination condition is.

- *Removing criteria*. If the LQE of a codeword is 0, this codeword will be removed directly. If the LQE of a codeword is the lowest, the codeword is denoted as a *loser*: the *loser* will be removed. The codeword adjacent to the *loser* (we call it neighbor of *loser*) accepts the Voronoi region of a *loser*. For example, in Fig. 1, the LQE of the 2nd codeword is 0; it will be removed. Then, the LQE of the 3rd codeword becomes the lowest; it will also be removed. If we remove the 3rd codeword, all vectors in the Voronoi region R_3 are assigned to Voronoi region R_4 and the 4th codeword is moved to the centroid of the new region.
- *Insertion criteria*. To insert a new codeword, we must avoid the bad situation shown by the 3rd codeword and the 4th codeword in Fig. 1. Therefore, a new codeword is inserted near the codeword with the highest LQE (*winner*). In Fig. 1, the 1st codeword is the *winner*; we randomly choose a vector that lies in the Voronoi region of the 1st codeword as a new codeword.
- *Termination condition*. We hope the removal–insertion of codewords is able to fine-tune the codebook. For a fixed number of codeword tasks, removal and subsequent insertion of codewords will engender a decrease in distortion error; for fixed distortion error tasks, removal and subsequent insertion of codewords will decrease the number of codewords without increasing the distortion error. Consequently, the termination condition will be:
  (1) For predefined number of codewords (Algorithm 3.1), if the removal–insertion process cannot engender a decrease in quantization error, stop.

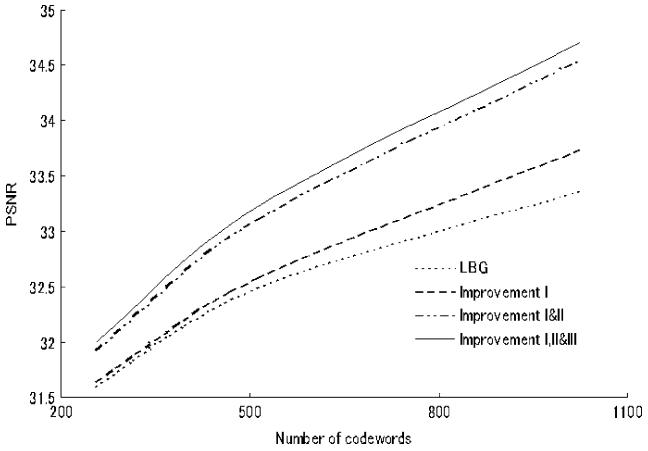

Fig. 4. Comparison results: LBG, Improvement I, Improvements I and II, and Improvements I, II and III (Improvements I, II and III indicate using the removal–insertion process to fine-tune the codebook generated by Improvements I and II).

(2) For a predefined distortion error threshold (Algorithm 3.2), if the removal–insertion process cannot lead to a decrease in the number of codewords, stop.

We use the removal–insertion process to fine-tune the codebook generated by Algorithm 3.1 with adaptive distance function, then compare the results with LBG, Algorithm 3.1 with the squared Euclidean distance (Improvement I), and Algorithm 3.1 with adaptive distance (Improvements I and II) in Fig. 4. It shows that the removal–insertion process fine-tunes the codebook very well; the reconstructed image quality is improved. We performed quintuple-testing for Improvement III, but the results remained almost identical, implying that the results obtained using the removal–insertion process become stable and independent of initial conditions.

3.4. Proposed algorithm

In the above sections and this section, some notations are used. Here we specify those acronyms and notations.

Notations

C	codebook
c_i	ith codeword of codebook C
E_i	Local Quantization Error (LQE) of codeword c_i
R_i	Voronoi region of codeword c_i
m	predefined number of codewords
η	predefined distortion error limitation
q	current number of codewords
ξ	current Mean Quantization Error (MQE)
iq	inherited number of codewords. It stores the number of codewords of the last iteration
$i\xi$	inherited MQE. It stores the MQE of last iteration
iC	inherited codebook. It stores the codebook of last iteration
winner	the codeword with largest LQE
loser	the codeword whose LQE is the lowest

With the above analysis, we give the proposed method in this section. Algorithm 3.3 gives the outline of proposed method.

Algorithm 3.3. Outline of adaptive incremental LBG

(1) Initialize codebook C to contain one codeword; define the adaptive distance function d.
(2) Execute the LBG algorithm (Algorithm 2.1) for codebook C.
(3) If the insertion condition is satisfied, insert a new codeword to codebook C, then go to step 2. Else, if the termination condition is satisfied, output the codebook and stop the algorithm; if the termination condition is not satisfied, execute step 4.
(4) Remove codewords with no contribution or the lowest contribution to reduction of distortion error, go to step 2.

In step 3 of Algorithm 3.3, for the predefined number of codewords task, the insertion condition pertains if the current codewords are fewer than a predefined number of codewords; for the predefined distortion error task, the insertion condition pertains if the current distortion error is larger than the predefined distortion error.

In Algorithm 3.3, if we execute LBG for the whole codebook after every codeword removal or insertion, the computation load will be very heavy. To avoid heavy computation load, we propose the following technique: after inserting a new codeword, we only execute LBG within the Voronoi region of *winner* for only two codewords (*winner* and the new codeword) and it will need very small computation load; for removal of codewords, we just combine the Voronoi region of removed codewords with the Voronoi region of adjacent codeword and do not need to execute LBG, no increase of computational load is incurred. In fact, the Voronoi regions of adjacent codewords will be influenced after inserting a new codeword or removing a codeword. The proposed technique limits the influence inside the Voronoi region of the *winner* or the *loser*. Even though this technique does not assure an optimal distribution of codewords, the experimental results in Section 4 demonstrate its validity.

With the above analysis, we give the detail of the proposed method in Algorithm 3.4, which is a modification of Algorithm 3.3 for a small computation load. Algorithm 3.4 will not have great computation load; it is suitable for large data sets or real world tasks.

Algorithm 3.4. Adaptive incremental LBG

(1) Initialize the codebook C to contain one codeword c_1

$$C = \{c_1\}, \tag{17}$$

with codeword c_1 chosen randomly from the original data vectors set. Predefine the total number of codewords as m (or predefine the distortion error threshold as η), give the definition of adaptive distance function d, initialize inherited numbers, inherited MQEs, and the inherited codebook as the following

$$iq = +\infty \tag{18}$$

$$i\xi = +\infty \tag{19}$$

$$iC = C \tag{20}$$

(2) With codebook C, execute the LBG algorithm (Algorithm 2.1) to optimize codebook C. Record the current number of codewords q, current MQE ξ, every LQE E_i, and Voronoi region R_i of c_i ($i=1,\ldots,q$).

(3) (a) If the current codewords q are fewer than m (or if the current MQE ξ is greater than η), insert a new codeword c_{new} to codebook C, i.e. if $q<m$ (or $\xi>\eta$), do the following to insert a new codeword:

- Find the *winner* whose LQE is largest.

$$C_{\text{winner}} = \underset{c_i \in C}{\operatorname{argmax}} E_i \tag{21}$$

- Randomly choose one vector c_{new} from the Voronoi region R_{winner} of c_{winner}.
- Add the c_{new} to codebook C.

$$C = C \cup c_{\text{new}} \tag{22}$$

- Execute LBG within R_{winner} region with two codewords: *winner* and c_{new}. Record the current q, MQE ξ, E_i, and R_i; then go to step 3.

(b) If the current number of codewords q is equal to m (or if the current MQE ξ is less than or equal to η), i.e. if $q=m$ (or $\eta \geq \xi$), do the following:

- If $\xi < i\xi$ (or $q < iq$)

$$i\xi = \xi \tag{23}$$

$$iq = q \tag{24}$$

$$iC = C \tag{25}$$

go to step 4.

- If $\xi \geq i\xi$ (or $q \geq iq$), output iC (codebook), iq (number of codewords), and $i\xi$ (MQE) are the final results, stop.

(4) Execute the following steps to remove codewords with 0 or lowest LQE.

- Find the codewords whose LQEs are 0, remove those codewords.
- Find the *loser*, whose LQE is the lowest:

$$c_{\text{loser}} = \underset{c_i \in C, E_i \neq 0}{\operatorname{argmin}} E_i. \tag{26}$$

- Delete c_{loser} from codebook C.

$$C = C \backslash c_{\text{loser}} \tag{27}$$

- Find the codeword c_a adjacent to c_{loser}, assign all vectors in R_{loser} to R_a, substitute c_a by the centroid of new Voronoi region.

$$R'_a = R_a \cup R_{\text{loser}} \tag{28}$$

$$c'_a = \frac{1}{N_a} \sum_{i=1}^{N_a} x_i, \quad x_i \in R'_a x \tag{29}$$

- Record current q, MQE ξ, E_i, and R_i; then go to step 3.

4. Experiment

This section presents an examination of the proposed algorithm using several image compression tasks. We compare our results with the recently published efficient algorithm ELBG (Patane & Russo, 2001). In those experiments, three images Lena (512×512×8) (Fig. 5), Gray21 (512×512×8) (Fig. 8), and Boat (512×512×8) (Fig. 9) are tested.

Three experiments are performed to test the different properties of the proposed method.

4.1. Experiment 1: predefining the number of codewords

In this experiment, we realize the traditional task of vector quantization, i.e. with a fixed number of codewords, generate a suitable codebook and maximize the PSNR (thereby minimizing the distortion error). The test image is Lena (512×512×8) (Fig. 5).

Patane and Russo (2001) proposed an Enhanced LBG (ELBG) to compress the Lena image. They compared their results with the Modified k-means method of Lee, Baek, and Sung (1997). Those results are listed in Table 6 of Patane and Russo (2001). According to that table (Patane & Russo, 2001), ELBG is tested with 256, 512, and 1024 codewords.

Fig. 5. Original image of Lena (512×512×8).

Table 1
Comparison results: with a fixed number of codewords, LBG, Modified k-means, ELBG, and the proposed method

Number of codewords	PSNR (dB)			
	LBG	Modified k-means	ELBG	Proposed
256	31.60	31.92	31.94	32.01
512	32.49	33.09	33.14	33.22
1024	33.37	34.42	34.59	34.71

The ELBG results are better than those achieved by Modified k-means (Lee et al., 1997). We also list the comparison results here in columns 3 and 4 of Table 1.

To test the proposed method, we also set the predefined number of codewords m as 256, 512, and 1024, then execute Algorithm 3.4 to generate a suitable codebook. After we get the codebook, we use the codebook to encode the original image and get the related symbol channels, then, at the decoder, we decode such symbol channels with associate codewords in a codebook; then we calculated the PSNR of the reconstructed image. Fig. 6 portrays a reconstructed image of Lena with 256 codewords. Table 1 presents a comparison of the proposed method with LBG, Modified k-means, and ELBG.

In Table 1, column 5 lists results of the proposed method. Compared with some other LBG-based methods (column 2 of LBG, column 3 of Modified k-means, column 4 of ELBG) with the same number of codewords, the proposed method gets highest PSNR, i.e. with the same compression ratio, the proposed method gets the best reconstruction image quality. We say that the proposed method provides the best codebook.

To demonstrate the efficiency of the proposed method, for the Lena image, we record the LQE of every codeword after the algorithm stopped. According to Gersho's theorem (Gersho, 1986), if the LQEs of all codewords are the same (equal to MQE), the quantizer is optimal. Enlightened by the 'utility index' of ELBG (Patane & Russo, 2001), to simplify the analysis, we define the 'Error Index' (EI) as:

$$\text{Error Index}_i = \frac{\text{LQE}_i}{n\text{MQE}/m}, \quad i = 1, \ldots, m \tag{30}$$

Using this definition, we know that, if all EI are equal to 1, the quantizer is optimal. A low EI means low LQE, therefore implying a low contribution to the reduction of distortion error; a large EI implies large LQE and a large contribution to the reduction of distortion error. We compare the distribution of EI of the proposed method with LBG and ELBG algorithms in Fig. 7; the number of codewords is 512 for all three algorithms.

Fig. 7(a) shows the EI distribution of LBG results, the EI distributed in [0,6.5] shows that the codewords of LBG make quite different contributions to the reduction of distortion error. Too many codewords exist whose EI is less than 1.0; some codewords have an EI that is greater than 6.0. The distribution shows that these results are not optimal, many codewords are assigned to small clusters and few codewords are assigned to large clusters. Therefore, plenty of codewords must be moved from the low distortion error region to the high distortion error region.

Fig. 7(b) shows the EI distribution of ELBG results, the EI distributed in [0,2.5], some codewords with low EI in Fig. 7(a) are moved to regions with high EI, and ELBG achieves a more balanced EI contribution than LBG; it makes ELBG work better than LBG.

Fig. 7(c) shows the EI distribution of the proposed method. The EI are distributed in [0,1.9] and the codewords with large LQE (EI is greater than or equal to 1) are more than the codewords with small LQE. It shows that the proposed method offers a more balanced contribution to the decreasing of distortion error than LBG and ELBG. Many codewords are assigned to large clusters; a few codewords are assigned to small clusters. It is impossible for us to get exactly the same LQE for all codewords (because the number of codewords is finite), the proposed method achieves a good distribution; moreover, it enhances the proposed method better than some previously published LBG-based methods.

Fig. 6. Reconstructed image of Fig. 5, $m = 256$, PSNR = 32.01 dB.

4.2. Experiment 2: predefining the PSNR threshold

This experiment realizes a task that is unsolved by other LBG-based methods: with a fixed PSNR threshold, minimize the number of codewords and generate a suitable codebook. The test image is Lena ($512 \times 512 \times 8$) (Fig. 5).

Using the ELBG algorithm for the Lena image, with 256, 512, and 1024 codewords, the resultant PSNRs are 31.94, 33.14, and 34.59 dB, respectively (Table 6 of reference Patane & Russo, 2001). Here, we set the PSNR of ELBG results as the PSNR threshold, and reasonably assume that to get 31.94, 33.14, or 34.59 dB PSNR, the ELBG algorithm needs 256, 512, or 1024 codewords, respectively; even ELBG cannot be used to solve this problem: given a PSNR threshold, minimize the number of codewords.

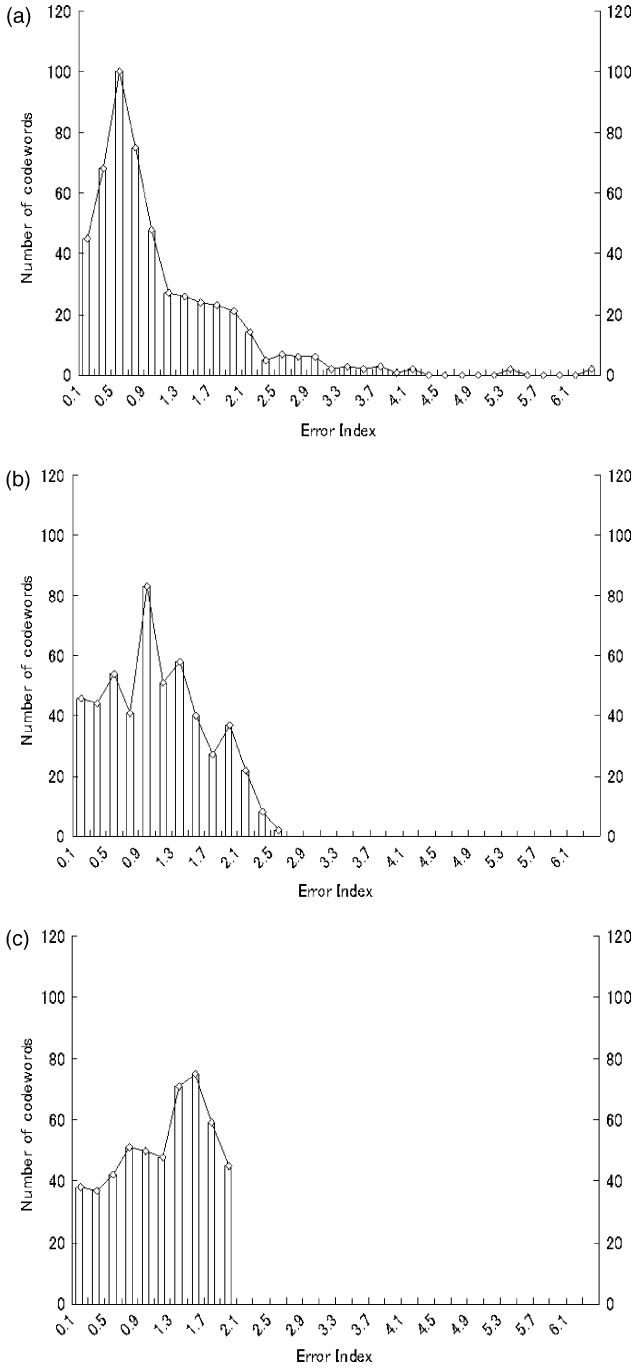

Fig. 7. Error Index distribution comparison: LBG, ELBG, and the proposed method, with 512 codewords. (a) Error Index distribution of LBG. (b) Error Index distribution of ELBG. (c) Error Index distribution of the proposed method.

Then, we set 31.94, 33.14, and 34.59 dB as the PSNR thresholds, and use Algorithm 3.4 to minimize the required number of codewords to thereby generate a codebook. Table 2 shows comparative results for ELBG and the proposed method.

The results obtained by the proposed method are listed in column 3 of Table 2. It shows that, with the same PSNR, the proposed method requires fewer codewords than ELBG. With the same reconstruction quality (PSNR), the proposed method obtains a higher compression ratio than ELBG.

Table 2
Comparison results: with fixed PSNR, ELBG and the proposed method

PSNR (dB)	Number of codewords	
	ELBG	Proposed
31.94	256	244
33.14	512	488
34.59	1024	988

4.3. Experiment 3: with the same PSNR threshold, encoding different images

In this experiment, we realize this task: using the same fixed PSNR threshold, find suitable codebooks for different images with different detail. The test images are Lena (512×512×8) (Fig. 5), Gray21 (512×512×8) (Fig. 8), and Boat (512×512×8) (Fig. 9). The three images have different details. For example, Gray21 is flat and little detail is visible. Lena has more detail than Gray21, but less detail than Boat.

One target of this experiment is to test if the proposed method works well for different images, not just works well for Lena image. Another target is to check with the same PSNR threshold, if different images with different details need different numbers of codewords. We set the PSNR thresholds as 28.0, 30.0, and 33.0 dB, then execute Algorithm 3.4 to minimize the number of codewords and find suitable codebooks for the three images.

Table 3 lists the results. For different images, if there is less detail in the image, fewer codewords are needed. Following the increase in detail, the number of codewords will also be increased.

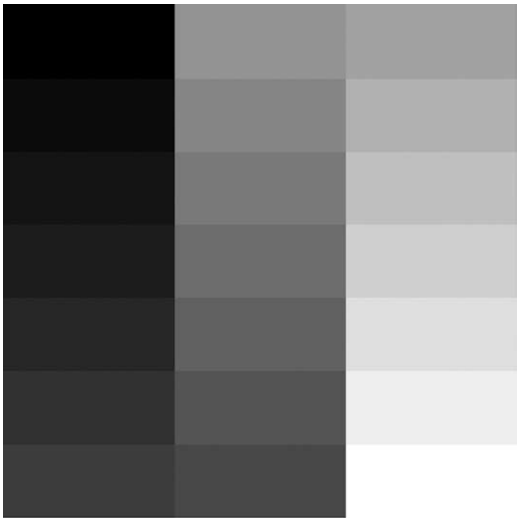

Fig. 8. Original image of Gray21 (512×512×8).

Fig. 9. Original image of Boat (512×512×8).

Table 3
With fixed PSNR, the proposed method works for different images

PSNR (dB)	Number of codewords		
	Gray21	Lena	Boat
28	9	22	54
30	12	76	199
33	15	454	1018

This experiment proves that the proposed method is useful to minimize the number of codewords. For different images, the number of codewords might be different. This conclusion will be very useful in some applications by which we need to encode different data set with the same distortion error. For example, if we attempt to set up an image database (which is composed of three images Lena, Gray21, and Boat) with the same PSNR (33.0 dB) for every image, with the traditional LBG-based methods (LBG, ELBG, etc.), we must encode every image with at least 1018 codewords to ensure that the reconstructed image quality can be satisfied for all images. In contrast, using the proposed method, we need only 15 codewords for Gray21, 454 codewords for Lena, and 1018 codewords for Boat. Thereby, the proposed method requires much less storage for this image database. If more images exist in the image database, much storage space will be saved through the use of the proposed method.

5. Conclusions

This paper presented an adaptive incremental LBG algorithm for vector quantization. New codewords are inserted into the codebook until the insertion condition is not satisfied. The adaptive distance function is adopted to improve the quantization process. The removal–insertion technique fine-tunes the codebook to make the proposed method independent of initial condition. The techniques achieve better performance than the majority of recently published methods. The proposed method not only works for traditional VQ tasks (predefine the number of codewords), it also works for new requirements (predefine the distortion error limitation). Experiments for complex image compression tasks also show the efficiency of the proposed method. Compared to the recently published method ELBG, the proposed method can provide a better codebook.

References

Chan, L. A., Nasrabadi, N. M., & Mirelli, V. (1996). Multi-stage target recognition using modular vector quantizers and multilayer perceptrons. *Proceedings of the CVPR'96* , 114–119.

Chinrungrueng, C., & Sequin, C. (1995). Optimal adaptive k-means algorithm with dynamic adjustment of learning rate. *IEEE Transaction on Neural Networks*, *8*(5), 729–743.

Cosman, P., Gray, R., & Vetterli, M. (1996). Vector quantization of image subbands: a survey. *IEEE Transactions on Image Processing*, *5*(2), 202–225.

Fritzke, B. (1995). A growing neural gas network learns topologies. In Advances in neural information processing systems (pp. 625–632).

Fritzke, B. (1997). The LBG-U method for vector quantization—an improvement over LBG inspired from neural networks. *Neural Processing Letters*, *5*(1).

Gersho, A. (1986). In E. Biglieri, & G. Prati (Eds.), *Digital communications*. Amsterdam: Elsevier/North-Holland.

Gray, R. M. (1984). Vector quantization. *IEEE ASSP Magazine*, *1*, 4–29.

Jolion, J., Meer, P., & Bataouche, S. (1991). Robust clustering with applications in computer vision. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, *13*, 791–802.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, *43*, 59–69.

Lee, D., Baek, S., & Sung, K. (1997). Modified k-means algorithm for vector quantizer design. *IEEE Signal Processing Letters*, *4*(1), 2–4.

Likas, A., Vlassis, N., & Verbeek, J. J. (2003). The global k-means clustering algorithm. *Pattern Recognition*, *36*, 451–461.

Linde, Y., Buzo, A., & Gray, R. M. (1980). An algorithm for vector quantizer design. *IEEE Transactions on Communication*, *COM-28*, 84–95.

Lloyd, S. P. (1957). Least squares quantization in PCM's. *Bell Telephone Laboratories Paper*, Murray Hill, NJ.

Martinetz, T. M., Berkovich, S. G., & Schulten, K. J. (1993). 'Neural-gas' network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, *4*(4), 558–569.

Max, J. (1960). Quantizing for minimum distortion. *IRE Transactions on Information Theory*, *IT-6*, 7–12.

Paliwal, K., & Atal, B. (1993). Efficient vector quantization of LPC parameters at 24 bits/frame. *IEEE Transactions on Speech and Audio Processing*, *1*(1), 3–14.

Patane, G., & Russo, M. (2001). The enhanced LBG algorithm. *Neural Networks*, *14*, 1219–1237.

Perlmutter, K., Perlmutter, S., Gray, R., Olshen, R., & Oehler, K. (1996). Bayes risk weighted vector quantization with posterior estimation for image compression and classification. *IEEE Transactions on Image Processing*, *5*(2), 347–360.

Vladimir, C., & Filip, M. (1997). *Learning from data—Concepts, theory, and methods*. New York: Wiley.