VECTOR QUANTIZATION CODE BOOK DESIGN USING NEURAL NETWORKS

by

Thomas M. Parks

Memorandum No. UCB/ERL M90/111

3 December 1990

VECTOR QUANTIZATION CODE BOOK DESIGN USING NEURAL NETWORKS

by

Thomas M. Parks

Memorandum No. UCB/ERL M90/111

3 December 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

VECTOR QUANTIZATION CODE BOOK DESIGN USING NEURAL NETWORKS

by

Thomas M. Parks

Memorandum No. UCB/ERL M90/111

3 December 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Vector Quantization
Code Book Design Using Neural Networks *

Thomas M. Parks

3 December 1990

Abstract

The Kohonen Self-Organizing Feature Map algorithm is compared to the K-Means vector quantization algorithm. Computation and storage requirements are calculated for both algorithms. A new algorithm which takes advantage of the structured code book produced by Kohonen's algorithm is introduced. This algorithm offers a significant computational savings over full-search vector quantization without imposing a storage cost penalty. The results of simulation studies are presented and the performance of the algorithms is compared.

1 Introduction

This paper evaluates the use of neural network algorithms for vector quantization. Section 2 gives a brief overview of vector quantization, followed by descriptions of the K-Means algorithm in Section 3 and the Kohonen Self-Organizing Feature Map in Section 4. Computation and storage costs of these two algorithms are compared in Section 5. Simulation results demonstrating the performance of the algorithms are presented in Section 6. Section 7 concludes with a discussion of issues that deserve additional consideration.

*This report has been sponsored in part by the Air Force Office of Scientific Research (AFOSR/JSEP) under Contract Number F49620-90-C-0029.

2 Vector Quantization

A review of vector quantization techniques appears in [MRG85]. Vector quantization involves the mapping of a continuous valued input vector $\vec{x}$ of dimension N to a discrete valued reconstruction vector $\vec{y}$ of the same dimension. The reconstruction vector typically can only take on a finite number of values, M. The set $\{\vec{y}_i, 1 \leq i \leq M\}$ of reconstruction vectors is referred to as the code book, and the values $\vec{y}_i$ are the code vectors. Vector quantization can also be thought of as a classification problem. The input $\vec{x}$ is classified into one of M classes C_i, and assigned the code vector $\vec{y}_i$ associated with that class.

The quantization error which results when $\vec{x}$ is coded as $\vec{y}$ can be quantified by a distortion measure, $d(\vec{x}, \vec{y})$. The mean-square error distortion measure, denoted by $d_2(\vec{x}, \vec{y})$, is given by the equation:

$$d_2(\vec{x}, \vec{y}) = \frac{1}{N}(\vec{x}-\vec{y})^T(\vec{x}-\vec{y}) = \frac{1}{N}\sum_{k=1}^{N}(x_k-y_k)^2 \quad (1)$$

Given a finite training sequence $\vec{x}(t)$, which is thought of as a random process, the goal in designing the code book is to minimize the

expected value of the overall distortion

$$E[D] = \sum_{i=1}^{M} P(\vec{x} \in C_i) \int_{x \in C_i} d(\vec{x}, \vec{y}_i) p(\vec{x}) d\vec{x} \tag{2}$$

where $P(\vec{x} \in C_i)$ is the probability that $\vec{x}$ belongs to class C_i, and $p(\vec{x})$ is the probability density function of $\vec{x}$.

For the case where $d(\vec{x}, \vec{y})$ is the mean-square error, the overall distortion is minimized when:

$$\vec{y}_i = \frac{1}{L_i} \sum_{x \in C_i} \vec{x}(t) \tag{3}$$

where L_i is the number of training vectors in C_i. In other words, the code vector $\vec{y}_i$ should be the centroid of the class C_i.

For scalar quantization, which is a special case of vector quantization, the coded outputs, or quantization levels y, can be chosen to have a non-uniform spacing to match the probability distribution of the input x and to minimize the overall distortion. The quantization levels are closer together in regions where x has a higher probability, and farther apart where x has a lower probability. This implies that the classes C_i associated with the quantization levels y_i are not of uniform size. They are, however, all the same shape: they are all segments of the real line.

In the more general vector quantization problem, several properties of the input $\vec{x}$ can be exploited: the dimensionality of the vector, linear dependence among the vector elements x_k, statistical dependence among the elements x_k, and the shape of the probability density function. In the case of scalar quantization, only the last property could be exploited. Dependencies among the vector elements restrict the possible values of $\vec{x}$, thus they can be exploited by limiting the possible reconstruction values $\vec{y}_i$. Vector dimensionality allows the classes C_i associated with the code vectors $\vec{y}_i$ to vary both in size *and* in shape.

3 K-Means Clustering Algorithm

The K-Means clustering algorithm is also known as the generalized Lloyd algorithm or the LBG algorithm [MRG85, LRCG89]. It is an iterative method for designing a code book which converges to a local minimum. After each iteration, it is necessary to perform a termination test. Typically this involves measuring the decrease in overall distortion.

1. Initialize code vectors by a suitable method.

2. Classify the entire training sequence $\vec{x}(t)$ into the classes C_i by the nearest neighbor rule:

$$\vec{x} \in C_j \Leftrightarrow d(\vec{x}, \vec{y}_j) \leq d(\vec{x}, \vec{y}_i), \forall i \neq j$$

3. Update code vectors for each class by computing the centroid:

$$\vec{y}_i = \frac{1}{L_i} \sum_{x \in C_i} \vec{x}(t)$$

4. Repeat by reclassifying training sequence using new code vectors.

The initialization step here is left intentionally vague. There are many methods for choosing an initial code book. Since the algorithm converges to a local minimum, one method of obtaining a more global minimum is to train using several different initial code books. Typically, the initial code book is either random, or is derived from a code book with $M - 1$ code vectors. For the case of $M = 1$, the code vector is just be the centroid of the training sequence. Thus multiple code books can be designed with increasing M, using the previous result for the initial code book.

4 Kohonen's Self Organizing Feature Map

The neural network algorithm described here is the feature map algorithm developed by Kohonen [KMS84, Lip87, Lip88]. The network consists of N input nodes, corresponding to the components x_k of the input vector $\vec{x}$, and M output nodes. The output nodes are arranged in a two-dimensional array, with neighboring nodes designed to have similar responses to the inputs. See Figure 1 for an illustration.

Each output node has connections to each of the inputs with weights. The weight for the connection between input node k and output node j is denoted by w_{kj}. The output of a node is calculated by taking the dot product of the input and the weights:

$$O_j = \vec{x}^T \vec{w}_j = \sum_{k=1}^{N} x_k w_{kj} \qquad (4)$$

The input is thus classified by the node with the maximum output. Maximizing the output value is equivalent to minimizing the mean-square error:

$$d_j = d_2(\vec{x}, \vec{w}_j) = \frac{1}{N} \sum_{k=1}^{N} (x_k - w_{kj})^2 \qquad (5)$$

Thus, the weight vectors $\vec{w}$ correspond to code vectors of a vector quantizer.

The training algorithm is usually performed in several phases, where different adaptation constants are used in each phase. Here is a brief description of the steps of the algorithm during phase i. Notice that when weights are updated, an entire neighborhood is updated. This forces nodes which are neighbors in the two-dimensional output array to have similar responses to the input, resulting in an ordered code book. This order can be exploited to save computation, as will be discussed in Section 5.

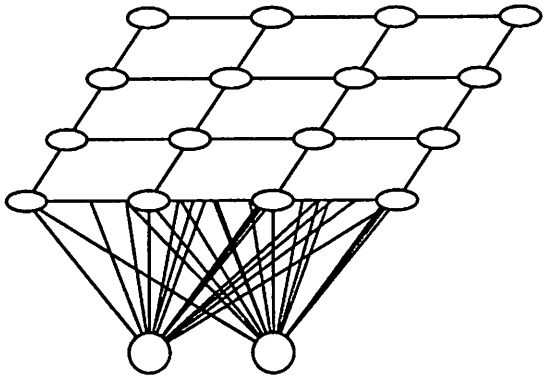

Figure 1: Feature Map with $N = 2$, $M = 16$

1. Initialize weights to small random values.
2. Present a new input $\vec{x}(t)$.
3. Compute distortion d_j for each output node.
4. Select the output node J with the minimum distortion.
5. Update weights to node J and its neighbors:

   $$\vec{w}_j(t+1) = \vec{w}_j(t) + \alpha(t)(\vec{x} - \vec{w}_j(t)), \forall \vec{w}_j \in N_J$$

6. Repeat by presenting the next input.

The neighborhood of node J is denoted by the set of nodes N_J. The adaptation gain $\alpha(t)$ is a function of time and distance:

$$\alpha(t) = K(t)\sigma^{r_{jJ}}(t) \qquad (6)$$

$$K(t) = K_i \frac{1 - (t - \tau_{i-1})}{\tau_i - \tau_{i-1}} \qquad (7)$$

$$\sigma(t) = \sigma_i \frac{1 - (t - \tau_{i-1})}{\tau_i - \tau_{i-1}} \qquad (8)$$

where K_i is the adaptation gain for phase i, σ_i is the distance weighting factor for phase i, r_{jJ} is the radius (or topological distance in the

output array) of the neighboring node j from the selected node J, and τ_i marks the end of training phase i.

When a neighborhood is updated, those code vectors which are topologically more distant from the selected node are changed less than those which are closer. The amount of adaptation decreases with time, allowing rapid adaptation at the beginning of training so that the code vectors can spread out to cover the N-dimensional space, and allowing slow adaptation later to provide a stable result. A more detailed description of the training process is given when experimental results are discussed in Section 6.

5 Algorithm Cost Comparison

A full search vector quantizer is computationally expensive, both while designing the code book during training, and while encoding vectors during normal operation. If the dimension of the input vectors is N and there are M code vectors, then a full search vector quantizer requires $O(M)$ distortion computations to compare an input vector to each of the code vectors. The number of storage locations required is $O(NM)$ to hold the code book, and $O(NL)$ to store the training sequence, where L is the number of training vectors. Typically $L \gg M$ to allow many training samples for each code vector, so storage of the training sequence is a significant cost. Both the Kohonen algorithm and the K-Means algorithm just described are full search quantizers. There are modifications which can be made to reduce the computational cost at the expense of increased storage or increased distortion.

5.1 Taking Advantage of an Ordered Code Book

The code book used in the Kohonen algorithm is arranged in a P-dimensional array, with $P = 2$ typically. As pointed out earlier, this code book has some degree of order since neighboring nodes are designed to have similar responses to inputs. Intuitively, one would expect the nearest neighbors of the node with minimum distortion to have smaller distortions than the remaining nodes. I call this type of order *neighborhood order*.

When the dimensionality of the code book matches the dimensionality of the data, $P = N$, I claim that in some cases the code book of the Kohonen algorithm takes on what I call *sorted order*. By this I mean that as one moves through the P-dimensional output array, away from the node with minimum distortion, distortion increases monotonically. If this were true, then a gradient search technique could be used. In general, when $P < N$, the code book does not achieve *sorted order*. This is because the distortion is inherently an N-dimensional function: it is a function of the N components of the code vectors. The code book maps this N-dimensional function onto a P-dimensional function of the coordinates of the code vectors in the output array. The mapping onto a lower-dimensional space may not preserve the monotonic properties of the distortion function. Exact characterization of the code book structure required by the gradient search algorithm, examination of conditions under which this structure can be constructed, and the design of a training algorithm which is guaranteed to produce this structure will be the subject of future research.

Both the decimated search and the gradient search algorithms, which are described below, reduce the computational cost of searching for the node with minimum distortion without increasing code book storage requirements. The

decimated search algorithm assumes that the code book has *neighborhood order*, but the number of required computations, while reduced, is still $O(M)$. The gradient search requires the more restrictive *sorted order*, and reduces the number of required computations to $O(\sqrt[P]{M})$.

5.1.1 Decimated Search

Truong and Mersereau [TM90] have proposed a *decimated search* algorithm which uses $\frac{M}{2^P}$ two-level search trees. For example, when $P = 2$, the distortion is first computed only for the nodes in the even numbered rows and columns of the output array. These nodes are the roots of the search trees. Then the distortion is computed for the neighbors of the root node with the minimum distortion. The neighboring nodes are the leaves of the search trees. This algorithm requires only $\frac{M}{2^P} + 3^P - 1$ distortion computations, however this is still $O(M)$.

5.1.2 Gradient Search

The search algorithm presented here assumes that the code book is in *sorted order*, and is essentially a gradient search algorithm.

1. Start at the center of the code book array.
2. Compute the distortion for the current code book vector and its nearest neighbors in the array.
3. Move to the code vector with the smallest distortion.
4. If there are any neighbors whose distortion has not been computed, then repeat by evaluating the distortions for the new neighbors.

The algorithm terminates when there are no new neighbors for which the distortion has not been computed. This can happen when the distortion of all the neighbors is higher than that of the current node (i.e. there are no new neighbors because we did not move to a neighbor in Step 3). It can also happen when we move to a node and discover that there are no new neighbors because we have reached the boundary of the output array.

The P-dimensional output array is arranged to be as square as possible, so at most $\sqrt[P]{M}$ steps are required to reach the node with minimum distortion. In this algorithm, a node has $2P$ neighbors, so at each step, at most $2P$ computations must be made, giving an upper bound of $2P\sqrt[P]{M}$ distortion computations. The number of distortion computations required has been reduced from $O(M)$ to $O(\sqrt[P]{M})$.

5.2 Using a Tree-Structured Code Book

A common way to reduce the computational cost of a full search vector quantizer is to use a tree-structured code book [MRG85, LRCG89]. Using the K-Means algorithm, for example, the training set is first divided into P classes using K-Means. Each of the resulting classes is then subdivided into P subclasses, again using K-Means. This is repeated until there is a total of M classes in the last level of the code book tree. To find the code vector with the minimum distortion, it is necessary to make P computations at each of $\log_P M$ levels, for a total of $P \log_P M$ distortion computations. This method reduces the computation cost from $O(M)$ to $O(\log_P M)$, but the required storage has increased. Previously, $O(NM)$ locations were required to store the code book. Using this method,

$$M \sum_{i=0}^{(\log_P M)-1} P^{-i} = \frac{M-1}{1-P^{-1}} \qquad (9)$$

code vectors must be stored. For $P = 2$, the storage requirements for the code book are doubled. However, this increase may be insignificant when compared to the storage requirements for the training sequence.

This brings up one great advantage that the Kohonen algorithm has over K-Means. Because the training vectors are presented sequentially in the Kohonen algorithm, there is no need to store the entire training set: training can be done on-line. This sequential presentation of the training vectors can cause problems, though, because the resulting code book depends on the order in which the training samples are presented.

6 Simulation Results

The Kohonen algorithm was implemented with three training phases. Phase 1 is intended to have large adaptation gain and large neighborhoods to allow very rapid adaptation. Phase 2 is intended to have moderate adaptation gain and fixed, small neighborhoods to allow slow adaptation. Phase 3 is intended to have small, fixed adaptation gain and small, fixed neighborhoods to demonstrate the stability of the adaptation.

During Phase 1, $0 < t \leq \tau_1$, the size of the neighborhood shrinks from including the entire array, to radius N_1. A radius of 1 would include only the 8 nearest neighbors, and a radius of 0 would not include any neighbors. The adaptation gain decays from K_1 to almost zero, and the distance term decreases from σ_1 to almost zero. During Phase 2, $\tau_1 < t \leq \tau_2$, the neighborhood radius remains constant at N_2, and the adaptation terms decay from K_2 and σ_2 to almost zero. During Phase 3, $\tau_2 < t \leq \tau_3$, the neighborhood radius remains constant at N_3, and the adaptation gain remains constant with terms K_3 and σ_3. Training ends at $t = \tau_3$.

6.1 2-Dimensional Random Data

6.1.1 Training

The first example used a two-dimensional training set of 4096 vectors with a uniform probability distribution; the code book contained 64 vectors. Both the full search and the gradient search versions of the Kohonen algorithm were simulated. The following constants were used for both versions:

$\tau_1 = 16384$	$\tau_2 = 32768$	$\tau_3 = 65536$
$N_1 = 1$	$N_2 = 1$	$N_3 = 0$
$K_1 = 1.0$	$K_2 = 0.1$	$K_3 = 0.01$
$\sigma_1 = 1.0$	$\sigma_2 = 0.1$	$\sigma_3 = 0.01$

For Phase 1, both K_1 and σ_1 are large, and the initial neighborhood radius is the width of the output array. This choice of values allows very rapid adaptation. For Phase 2, K_2 and σ_2 are each an order of magnitude smaller than the values used in Phase 1; the neighborhood radius is constant and includes only the nearest neighbors. This choice of values allows gradual adaptation. For Phase 3, K_3 and σ_3 are each an order of magnitude smaller than the values used in Phase 2; the neighborhood radius is constant and includes no neighbors. This choice of values allows very slow adaptation, and output nodes are allowed to adapt independently.

The time constants τ_1, τ_2, and τ_3 were chosen so that during Phase 1 and Phase 2, there would be approximately 256 training vectors for each code vector in each phase.

During training, a copy of the code book was made after each time the 4096 training vectors had been presented. This resulted in a sequence of 16 evolving code books. Some of the code books for the full search Kohonen algorithm are illustrated in Figures 2—6. The corresponding code books for the gradient search algorithm are shown in Figures 7—11. In the plots, lines are drawn connecting

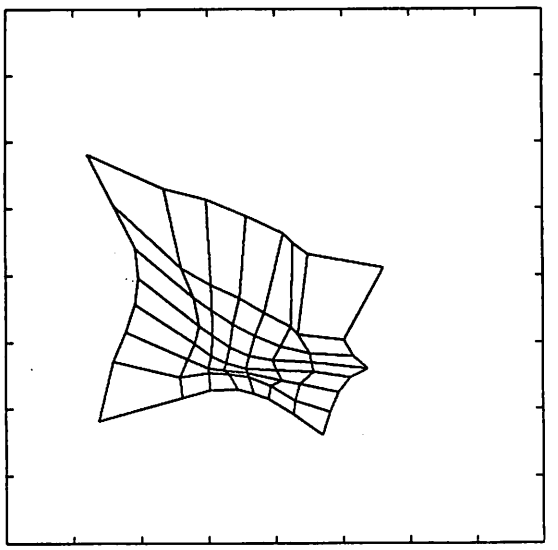

Figure 2: Full Search Code Book: t = 4096, SNR = 16.96 dB

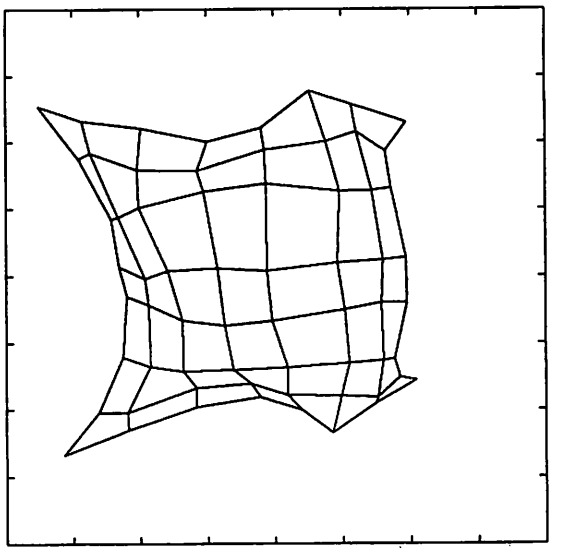

Figure 3: Full Search Code Book: t = 8192, SNR = 21.19 dB

each code vector to its topological neighbors in the output array in order to demonstrate the structure of the code book. If the code book is in *sorted order*, then neighbors in the P-dimensional output array will be neighbors in the N-dimensional vector space. If the code book is not in *sorted order*, then some lines drawn in the figures will cross. Figure 12 illustrates the *unsorted* code book produced by the K-Means algorithm.

Note that the code book spreads out more slowly in the gradient search version of the Kohonen algorithm. This occurs because the initial code book is not in *sorted order*, thus the selected node is not always the one with the minimum distortion. Once the code book is in *sorted order*, the gradient search selects the correct node.

6.1.2 Measurements

Using each of the 16 code books, the training sequence and a separate test sequence were

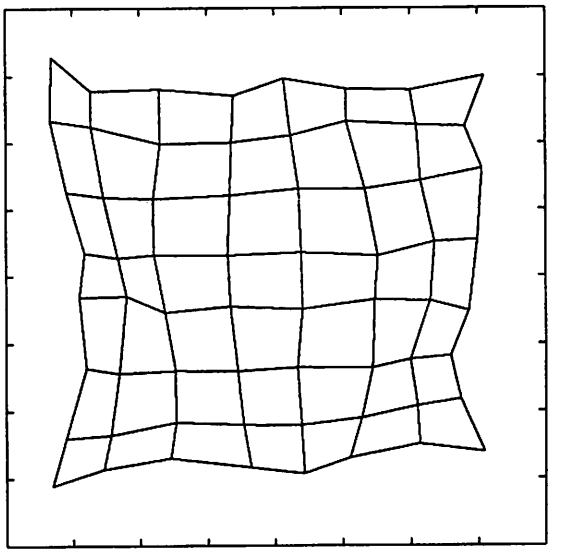

Figure 4: Full Search Code Book: t = 12288, SNR = 26.13 dB

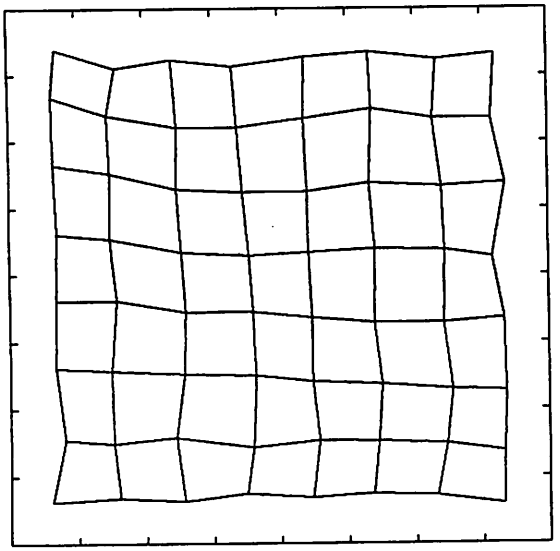

Figure 5: Full Search Code Book: t = 16384, SNR = 28.53 dB

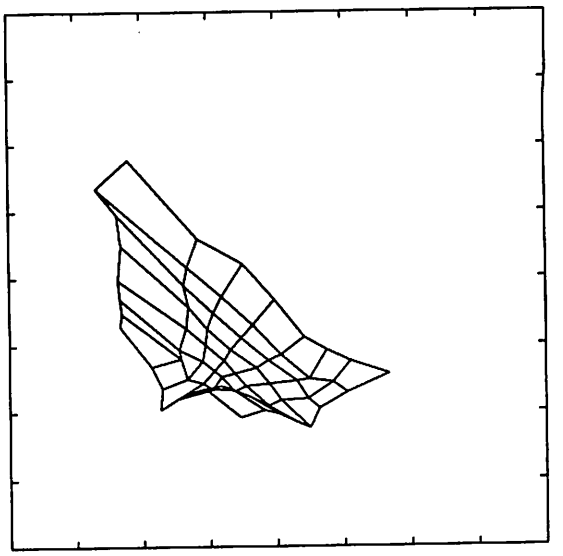

Figure 7: Gradient Search Code Book: t = 4096, SNR = 15.10 dB

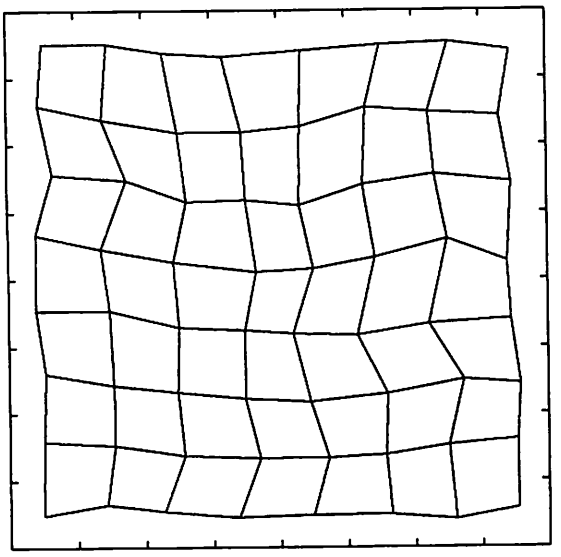

Figure 6: Full Search Code Book: t = 65536, SNR = 29.10 dB

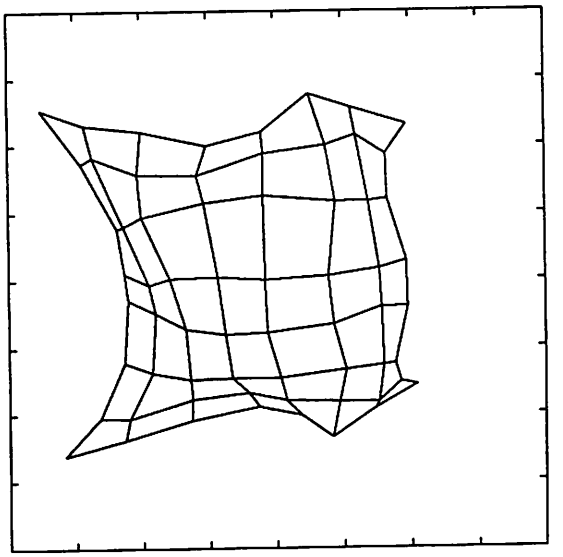

Figure 8: Gradient Search Code Book: t = 8192, SNR = 21.18 dB

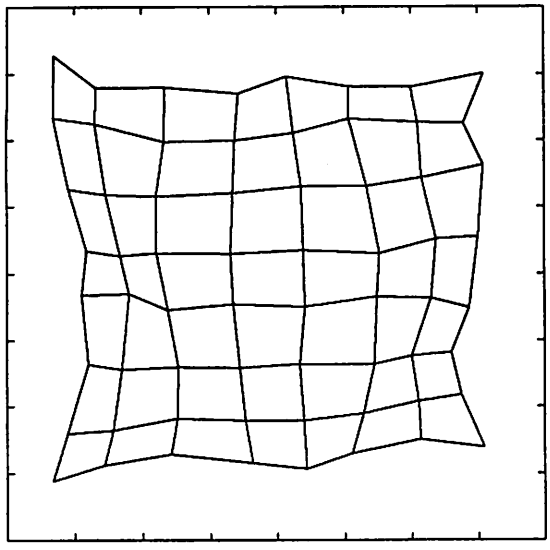

Figure 9: Gradient Search Code Book: t = 12288, SNR = 26.15 dB

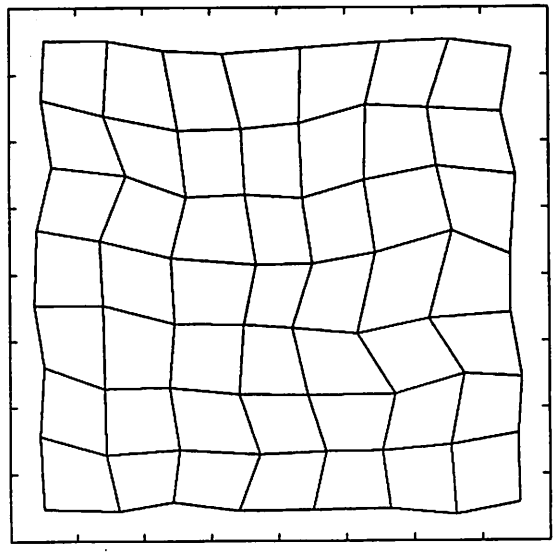

Figure 11: Gradient Search Code Book: t = 65536, SNR = 29.05 dB

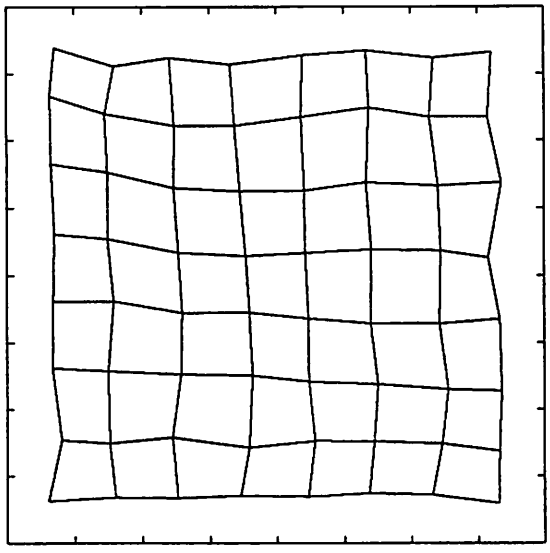

Figure 10: Gradient Search Code Book: t = 16384, SNR = 28.53 dB

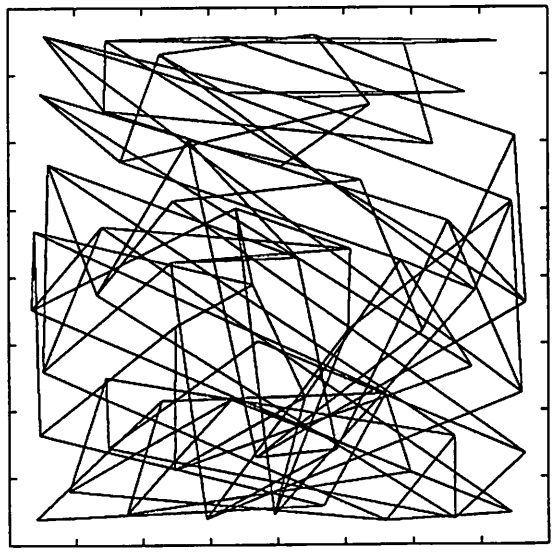

Figure 12: Unsorted K-Means Code Book

	Training Set	Test Set
Uniform	28.84	28.83
K-Means	29.12	28.63
Kohonen (full)	29.10	28.74
Kohonen (gradient)	29.05	28.70

Table 1: Distortion Measurements

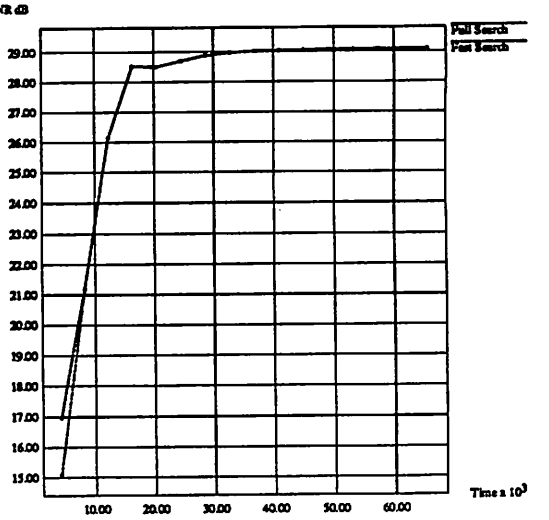

Figure 13: Distortion During Training

each coded and distortion measurements were made. Figure 13 shows how the distortion, or signal to noise ratio (SNR), evolved during training for both the full search and gradient search versions of the Kohonen algorithm. The gradual approach to the optimum SNR during Phase 3 indicates that the Kohonen algorithm is stable and does converge. The K-Means algorithm required 196 passes through the training sequence, but the Kohonen algorithm required only 16 passes. In fact, after only 4 passes through the training sequence, the Kohonen algorithm had achieved a SNR within 0.6 dB of the final value.

Table 1 compares the SNR for a uniform quantizer, and for the K-Means and Kohonen algorithms on both the training and test sequences. Compare these values to the theoretically optimum SNR of 28.85 dB for a uniform quantizer. Note that because the training and test sets are such small samples of the underlying random process, this optimum is not achieved by the uniform quantizer.

These distortion measurements were calculated as follows:

$$\overline{D} = \frac{1}{L}\sum_{t=1}^{L}\left(\frac{1}{N}\sum_{k=1}^{N}(x_k(t) - w_{kJ_t})^2\right) \quad (10)$$

where L is the number of input vectors, and J_t is the index of the output node with minimum distortion for the input $\vec{x}(t)$. Equation 10 is the expression for the average overall distortion. The signal to noise ratio is:

$$SNR = 10\log_{10} 1/\overline{D} \quad (11)$$

Notice that the final distortion measurement for the Kohonen algorithm is approximately the same as that obtained with K-Means. Because the training and test sequences were only small samples from a uniform random process, they did not have a perfectly uniform distribution. This explains the relatively poor behavior of the uniform code book: both vector quantization algorithms produced code books which reflected the actual distribution of the training sequence.

6.2 16-Dimensional Image Data

6.2.1 Training

The second example used the 256 × 256 image "girl" and broke it up into 4 × 4 blocks to form 4096 16-dimensional training vectors. The code book contained 256 vectors, for an aver-

	Training Set
K-Means	31.48
Kohonen (full)	30.64

Table 2: Distortion Measurements

age of 0.5 bits per pixel. Because the dimensionality of the output array was lower than that of the data, $P < N$, the code book did not achieve *sorted order* and the gradient search version of the Kohonen algorithm exhibited convergence problems. The results presented here are for only the full search version of the Kohonen algorithm and the K-Means algorithm. The following constants were used for training:

$$\begin{array}{lll} \tau_1 = 65536 & \tau_2 = 131072 & \tau_3 = 262144 \\ N_1 = 1 & N_2 = 1 & N_3 = 0 \\ K_1 = 1.0 & K_2 = 0.1 & K_3 = 0.01 \\ \sigma_1 = 1.0 & \sigma_2 = 0.1 & \sigma_3 = 0.01 \end{array}$$

6.2.2 Measurements

Figure 14 shows how the SNR evolved during training for the full search Kohonen algorithm. The K-Means algorithm required 328 passes through the training sequence, but the Kohonen algorithm required only 64 passes. After only 16 passes, the Kohonen algorithm had achieved a SNR within 0.8 dB of the final value. Figures 15–17 show the original image, and those coded with the K-Means and Kohonen algorithms. Table 2 compares the SNR for the K-Means and Kohonen algorithms. Keep in mind that these are distortion measurements for the training set only; there is no separate test set.

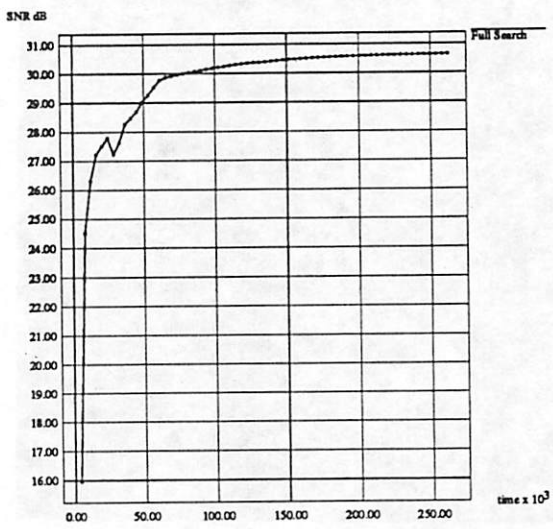

Figure 14: Distortion During Training

Figure 15: Original Image: 8 bits/pixel

Figure 16: K-Means Coding: 0.5 bits/pixel, SNR = 31.48 dB

Figure 17: Kohonen Coding: 0.5 bits/pixel, SNR = 30.64 dB

7 Conclusion

The Kohonen algorithm does converge, as can be seen in the examples presented in Section 6. During Phase 3 the adaptation gain is held constant instead of allowing it to decay to zero. To keep the distortion low, it is necessary to have a small adaptation gain, otherwise each new input would produce a large perturbation. A small adaptation gain, however, increases the training time. A balance between training time and distortion must be found on a case by case basis.

I introduced a method of taking advantage of the code book structure. In cases where $P = N$, as in the first example, it works well and seems not to have any convergence problems. In the second example where $P < N$, the code book did not exhibit the *sorted order* required by the search algorithm. For this case, either the dimensionality of the output array could have been increased, or the algorithm proposed by Truong and Mersereau could have been used. Either of these search methods could be used with a code book which was designed using K-Means if the code book were processed off-line after training to impose the required order.

Tree-structured code books degrade performance. This is because classes are split without regard to the number of training samples which belong to them. Thus it is possible to form subclasses which have few or even no training samples associated with them. Pruning away such classes from the tree can reduce the code book size with only a minor increase in distortion. The methods used by Gray and Riskin [LRCG89] require the design of a full tree-structured code book, followed by a pruning process which trades distortion for code book size.

If neural network techniques could be used to design a tree-structured code book, then the same pruning methods could be applied

off-line after training was complete. One possible way to design such a code book would be to first set up the tree structure and initialize the code vectors at all levels to random values. Then adapt the nodes at all levels of the code book as the inputs are presented. At each level, the node with the minimum distortion would be selected. Then the weights for that node and its neighbors would be updated and the same input would then be passed to the next level. It would be interesting to examine the convergence properties of this technique.

In conclusion, although the K-Means algorithm is optimal, it requires off-line training where the entire training set is available in storage. This precludes using K-Means in adaptive vector quantization applications. The Kohonen algorithm is inherently adaptive and requires no storage for the training set; it can be trained on-line. In a process where the underlying probability distribution changes with time, such as video images, the Kohonen algorithm would clearly be preferred. The simple computations required for the Kohonen algorithm also make it a viable candidate for VLSI implementation. Apparently the choice between K-Means and Kohonen depends on the specific application and hardware resources. Thus one is not superior to the other, only more appropriate for certain applications.

Acknowledgments

I would like to thank Dr. Avideh Zakhor for her encouragement and suggestions. She also provided the programs which I used to simulate the K-Means algorithm.

References

[KMS84] T. Kohonen, K. Makisara, and T. Saramaki. Phonotopic maps: Insightful representation of phonological features for speech recognition. In *IEEE Seventh International Conference on Pattern Recognition*, August 1984.

[Lip87] R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, April 1987.

[Lip88] R. P. Lippmann. Neural network classifiers for speech recognition. *The Lincoln Laboratory Journal*, 1(1), 1988.

[LRCG89] T. Lookabaugh, E.A. Riskin, P.A. Chou, and R.M. Gray. Variable rate vector quantization for speech, image, and video compression. Unpublished, April 1989.

[MRG85] J. Makhoul, S. Roucos, and H. Gish. Vector quantization in speech coding. *Proceedings of the IEEE*, 73(11), November 1985.

[TM90] K. Truong and R. Mersereau. Structural image codebooks and the self-organizing feature map algorithm. In *ICASSP*, 1990.