EXHIBIT M

TO DECLARATION OF S. MERRILL WEISS IN SUPPORT OF PLAINTIFF ACACIA MEDIA TECHNOLOGIES CORPORATION'S MEMORANDUM OF POINTS AND AUTHORITIES IN OPPOSITION TO ROUND 3 DEFENDANTS' MOTION FOR SUMMARY JUDGMENT OF INVALIDITY UNDER 35 U.S.C. § 112 OF THE '992, '863, AND '702 PATENTS; AND SATELLITE DEFENDANTS' MOTION FOR SUMMARY JUDGMENT OF INVALIDITY OF THE '992, '863, AND '720 PATENTS

Neural Networks for Vector Quantization of Speech and Images

ASHOK K. KRISHNAMURTHY, MEMBER, IEEE, STANLEY C. AHALT, MEMBER, IEEE, DOUGLAS E. MELTON, AND PRAKOON CHEN

***Abstract*—In this paper, we describe the use of neural networks for vector quantization (VQ). We first show how a collection of neural units can be used efficiently for VQ encoding, with the neural units performing the bulk of the computation in parallel. We then describe two *unsupervised* neural network learning algorithms for *training* the vector quantizer. A powerful feature of these new training algorithms is that the VQ codewords are determined in an adaptive manner, as compared to the popular LBG training algorithm [1], which requires that the entire training data be processed in a batch mode. The neural network approach allows for the possibility of training the vector quantizer on-line, thus adapting to the changing statistics of the input data. Finally, we compare the neural network VQ algorithms to the LBG algorithm in encoding a large database of speech signals and in encoding images.**

I. Introduction

There is a great deal of interest in the low bit rate coding of speech and images, and vector quantization (VQ) has emerged in recent years as a powerful technique that can provide large reductions in bit rate while preserving the essential signal characteristics [2], [3]. The most popular algorithm for VQ codebook design has been the so-called LBG or generalized Lloyd algorithm [1]. While the LBG algorithms and its variants have been widely studied [2], [4], the practical application of VQ has been limited because of the prohibitive amounts of computation associated with both the vector encoding and the codebook design stages [5].

The past few years have also seen a resurgence of interest in neural networks [6], and their application to a wide range of problems in clustering and pattern recognition [7]-[9]. One class of neural network structures, called self-organizing or competitive learning networks [10], [8], appears to be particularly suited for VQ, and a number of studies has been reported on using neural networks for VQ [11]-[16]. The use of neural networks for vector quantization has a number of significant advantages. First, neural networks are a highly parallel computer architecture and, thus, offer the potential for real-time VQ. Second, the large body of training techniques for neural networks can be adapted to yield new, and possibly better, algorithms for VQ codebook design. Finally, most neural network training algorithms are adaptive, and thus, neural network based VQ design algorithms can be used to build adaptive vector quantizers [17]. This is crucial in applications where the source statistics are changing over time.

Manuscript received October 2, 1989; revised January 24, 1990. This work was supported in part by a grant from Cray Research, Inc.

A. K. Krishnamurthy and S. C. Ahalt are with the Department of Electrical Engineering, The Ohio State University, Columbus, OH 43210.

D. E. Melton is with the Department of Electrical Engineering, University of Wisconsin, Madison, WI 53706.

P. Chen is with the Technical Staff, Oracle Corporation, Belmont, CA 94002.

IEEE Log Number 9036209.

This paper describes the use of neural networks for VQ. We first show how a collection of neural units can be used efficiently for VQ encoding, with the neural units performing the bulk of the computation in parallel. We then describe two *unsupervised* neural network learning algorithms for *training* the vector quantizer. These training algorithms are adaptive, in the sense that they process the training data one vector at a time, as compared to the LBG training algorithm [1], which requires that the entire training data be processed in a batch mode. Finally, we compare the performance of the neural network VQ design algorithms with the LBG algorithm in encoding a large database of speech signals and in encoding images.

II. Neural Networks for Vector Quantization

A. Vector Quantization

Vector quantization is a statistical method of encoding data for transmission to a receiver. In VQ, a k-dimensional data vector x that is to be encoded is represented as one of a finite set of M symbols. Associated with each symbol is a k-dimensional vector c_i called a codeword. The complete set of M codewords is called the codebook [2]. The k values in the vector can be, for example, successive sample values of a signal, or parameters extracted from the signal (such as linear predictive coding parameters [18]). The codebook $C = \{c_i, i = 1, \cdots, M\}$ is usually obtained through a training process using a large set of training data that is typical of the data that will be encountered in practice.

During the encoding process, each k-dimensional vector x that is to be encoded is compared to each of the M codewords in the codebook, and the distortion $d(x, c_i)$, $i = 1, \cdots, M$ between the input vector and the codeword is computed. The input vector x is then encoded as the *index* j of the codeword that yields the minimum distortion. The receiver, which is assumed to have a copy of the codebook, uses this index to look up the corresponding codeword c_j. Codeword c_j is then used as the encoded

Exhibit M Page 222

value of the vector x. Typically, $M = 2^L$ and the rate of the VQ is L/k bits per vector dimension. Note that the primary computational burden during the encoding process is that of computing the distortion between the input vector and each of the M codewords.

B. Neural Networks

Neural networks are highly parallel computing structures consisting of a number of simple processing units, called neural units, and a set of interconnections between these units and the inputs to the network. The inputs to any neural unit can be regarded as a vector x. Also associated with the neural unit is a weight vector w, and an activation function $g(w, x)$ that is the output produced by the unit. Thus, the overall input–output function of any neural unit is determined by its weight vector, its input, and the activation function of the unit.

In order to perform a particular task, neural nets undergo a training process in which the weight vectors associated with the units are modified. This process depends on the networks being presented with statistically representative data during the training process; the networks modify the weight vectors based on internally calculated error measures derived from the training data. There are a number of techniques used to modify the weights [6], but all of the techniques can be classified as either *supervised* or *unsupervised*.

Supervised training algorithms use training data that are labeled with the desired network output. An error measure is calculated from the desired output and the network-calculated output, and the weight vectors are consequently modified according to the supervised learning rule being used. One commonly used supervised learning rule is referred to as the *backpropagation training algorithm* [9], which is a generalization of the LMS algorithm. Supervised algorithms have been extensively used in a number of applications, including functional mapping, plant modeling, classification, and data abstraction, but have not been widely used for VQ.

Unsupervised training algorithms depend upon internally generated error measures which are derived solely from the training data. The network has no indication of the correct answer during training and, consequently, must derive the error and the necessary weight modifications directly from the statistics of the training data. In most cases, unsupervised training algorithms attempt to "cluster" or average portions of the training data into representative groups. Competitive learning algorithms are a class of such unsupervised training algorithms and they appear to be well-suited for use in VQ applications. We discuss the application of two such competitive learning algorithms, Kohonen's self-organizing feature maps (KSFM) [8] and frequency-sensitive competitive learning (FSCL) [16], [19], for VQ later in this paper.

It is straightforward to formulate a neural network structure for the encoding step in VQ [20], [21] and an example is shown in Fig. 1. Consider a neural network with M neural units, and associate with neural unit i a weight vector that is the ith codeword, i.e., $w_i = c_i$. Given any vector x that is to be encoded, x is fed in parallel to all the M neural units. Each of the units computes the distortion $d(x, w_i)$ between the input and its weight vector, i.e., the activation function $g(x, w_i) = d(x, w_i)$. The unit with the minimum distortion is called the "winning" neural unit, and the input vector x is encoded as the index of the winning unit. All computations, except for determining the winning neural unit, are carried out in parallel.

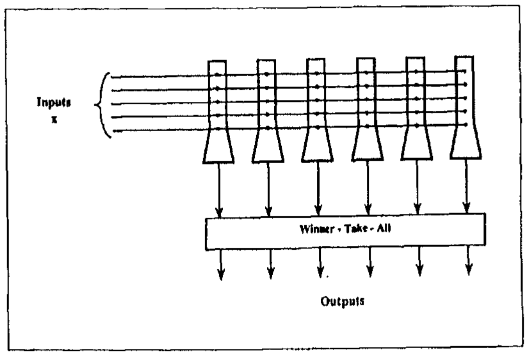


Fig. 1. Neural network structure for vector quantization.

The process of determining the winner can also be done using neural networks. One of the first neural network solutions to the winner-take-all problem was proposed by Grossberg [22]. Another solution is offered by Hecht-Nielsen [21] who proposes adding another layer of neural units at the output of the first layer. This second layer of units is trained to directly output the index of the winning unit. Lippmann [6] describes neural network structures that allow the winning unit to be determined using a binary search procedure. Finally, Winters and Rose [23] have shown that in VLSI implementations it is possible to determine the winning unit in $log_2 (M)$ steps.

III. Neural Network Training Algorithms for VQ Codebook Design

One of the major advantages of formulating the VQ problem in terms of neural networks is that the large body of neural network learning methods can be applied to the VQ task. Most of the neural network learning methods are adaptive; consequently, the VQ training algorithms derived from these are also adaptive and allow for the possibility of training the VQ codebook on line. We describe below two possible training algorithms: the Kohonen self-organizing feature map (KSFM) [8] and the frequency-sensitive competitive learning method (FSCL) [19].

Both KSFM and FSCL belong to the class of competitive learning (CL) training algorithms. In these algorithms, the weight vectors (or codewords) w_i associated with the neural units are initialized to small random values, and the algorithm iterates through the training data a number of times, adjusting w_i after the presentation of each input vector x. The simple competitive learning algorithm can be described as follows.

CL Training Algorithm:

1) Apply an input vector $\boldsymbol{x}$.
2) Find the distortion $D_i = d(\boldsymbol{x}, \boldsymbol{w}_i)$ for all output neural units.
3) Select the output unit with the smallest distortion and label it as the *winner* and its weight vector as $\boldsymbol{w}_{i*}$.
4) Adjust the selected weight vector

$$\boldsymbol{w}_{i*}(n+1) = \boldsymbol{w}_{i*}(n) + \epsilon(n)[\boldsymbol{x}(n) - \boldsymbol{w}_{i*}(n)] \tag{1}$$

where n is the training time index.

5) Repeat Steps 1) through 4) for all training vectors.

Note that the value selected for $\epsilon(n)$ does not depend upon the magnitude of the data. The training rule moves the weight toward the training vector by some fractional amount, $\epsilon(n)$. Typically, $0 < \epsilon(n) < 1$ and decreases as training progresses.

A major problem with the basic CL algorithm is that all the neural units are not equally utilized in representing the input data. In fact, it is typically the case that a few units do the bulk of the representation leading to unduly high distortions. The next two CL algorithms that we discuss overcome this underutilization problem, albeit in different ways.

A. The Kohonen Self-Organizing Feature Map

The Kohonen self-organizing feature map (KSFM) is a CL network that was proposed by Kohonen as a model for the process by which topological feature maps form in the brain [20]. In the KSFM network, each neural unit has an associated topological neighborhood of other neural units. During the training process, both the winning neural unit as well as the neural units in the neighborhood of the winner is updated. The size of the neighborhood is decreased as training progresses until each neighborhood has only one unit, i.e., the KSFM net becomes a CL net after sufficient training. By the use of neighborhoods, the KSFM network overcomes the problem of underutilized nodes discussed previously.

The neighborhood of a unit in KSFM is defined by means of a set of connections imposed on the neural units. For example, the connections might form a linear array or a rectangular grid. Typically, the neighborhoods are the set of units that are less than K connection lengths from the winning unit. Upon the receipt of a training element, a "winning" unit is selected. The weight vector of each unit in the winning unit's neighborhood is then updated as follows:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \epsilon(n, \mathfrak{D})[\boldsymbol{x}(n) - \boldsymbol{w}(n)]. \tag{2}$$

The training rule is similar to the one found in the CL net except that the learning rate is a function of distance $\mathfrak{D}$ from the selected unit as well as a function of training time n. Generally, $\epsilon(n, \mathfrak{D})$ decreases as the distance from the selected unit increases. Also as training progresses, the size of the neighborhood decreases in size.

The KSFM training algorithm may be formally stated as below; initially $\mathfrak{D}_{\max}$ is chosen large enough that all neural units are in the same neighborhood.

KSFM Training Algorithm:

1) Apply an input vector $\boldsymbol{x}$.
2) Find the distortion $D_i = d(\boldsymbol{x}, \boldsymbol{w}_i)$ for all output units.
3) Select the output unit $i*$ with the smallest distortion and label it as the *winning* unit.
4) Adjust the selected weight vector and its neighborhood of units

$$\boldsymbol{w}_{i*}(n+1) = \boldsymbol{w}_{i*}(n) + \epsilon(n, 0)[\boldsymbol{x}(n) - \boldsymbol{w}_{i*}(n)]. \tag{3}$$

For all units that are less than $\mathfrak{D}_{\max}$ away from $i*$

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \epsilon(n, \mathfrak{D})[\boldsymbol{x}(n) - \boldsymbol{w}(n)] \tag{4}$$

where n is the training time and $0 < \epsilon(n, \mathfrak{D}) < 1$.

5) Periodically decrease the extent of the neighborhood by decreasing $\mathfrak{D}_{\max}$ until $\mathfrak{D}_{\max} = 0$.
6) Repeat Steps 1) through 5) for all training vectors.

Figs. 2 and 3 show an example of the formation of the topological map for a simple case. The training set consisted of 1000 samples from a uniform density function in the region $R = \{-1 \leq x \leq 1, -1 \leq y \leq 1\}$. We show the movement of the codewords for two different neighborhood connections, a linear arrangement, and an arrangement in a two-dimensional grid. The movement of the codewords during training for the linear arrangement is shown in Fig. 2. This figure shows that by the third pass through the training file, the codewords have ordered themselves in a line. As training progresses, the codewords spread over the training space. After 25 iterations through the training set, the codebook adequately covers the space. In each iteration shown in Fig. 2, codewords are ordered as defined by the line of connections. Codeword 0 is adjacent to codeword 1, codeword 1 is adjacent to codeword 2, etc. A mathematical study of the ordering phenomenon is found in [8].

The movement of the codewords for the rectangular grid connection is shown in Fig. 3. Again, as the codewords cover the training space, they remain ordered as defined by the set of connections. Kohonen notes that it is sometimes advantageous to order the codewords depending upon the application [8].

The topological ordering imposed on the neural units in the KSFM structure has been used to advantage by a number of researchers. For example, Bradburn [24] uses the ordering to reduce the effects of transmission errors in a communication system; Lee and Peterson [17] use it to define a structurally adaptive vector quantizer that can add or kill neurons based on the statistics of the input data.

B. The Frequency-Sensitive Competitive Learning Network

The frequency-sensitive competitive learning (FSCL) net directly addresses the problem of neural unit under-

Exhibit M Page 224

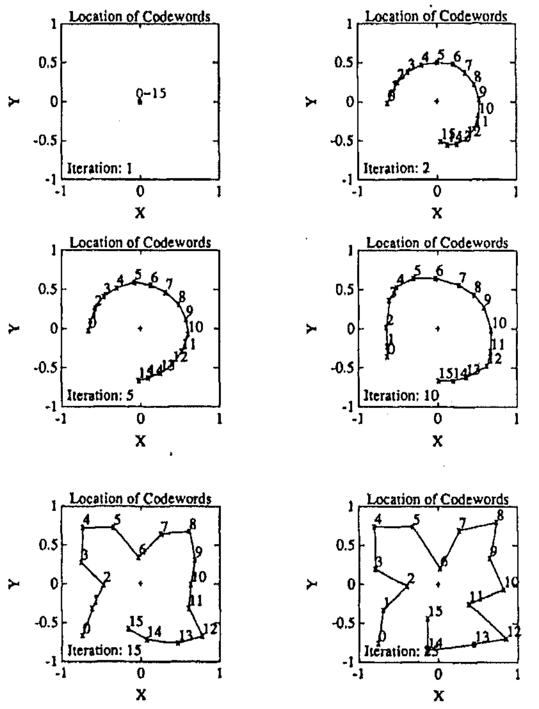


Fig. 2. Codeword movement, Kohonen self-organizing feature map, linear neighborhood.

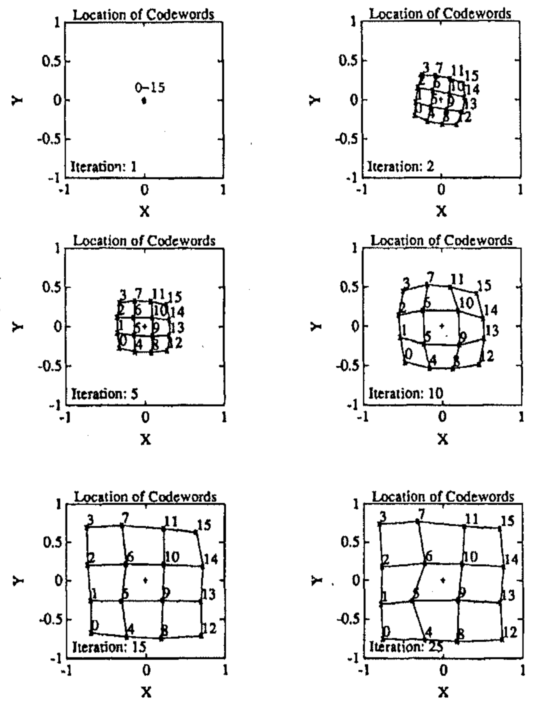


Fig. 3. Codeword movement, Kohonen self-organizing feature map, rectangular neighborhood.

utilization with the CL network by incorporating a measure of how frequently a unit has been the winner in the winner selection process [19]. Each neural unit maintains a count u_i of the number of times it was the winner node. The distortion measure that is used for selecting the "winning" node is now modified to be $\mathfrak{F}(u_i)d(x, w_i)$, where $\mathfrak{F}$ is a nondecreasing function called the "fairness" function. The fairness function is essentially a way of introducing a count-dependent weighting to the distortion measure. The effect of introducing this weighting is that if a neural unit "wins" frequently, its count increases and, consequently, its modified distortion measure also increases. This gives other units a chance to win the competition. Note that this is nothing but an implementation of Grossberg's conscience principle [22], and is similar to the conscience method of DeSieno [25].

The FSCL training algorithm can thus be stated as follows; the update counters u_i are initialized to be zero.

FSCL Training Algorithm

1) Apply an input vector x.
2) Find the distortion $D_i = \mathfrak{F}(u_i)d(x, w_i)$ for all output units where the u_i are the update counters.
3) Select the output unit i^* with the smallest distortion and label it as the *winning* unit and increment u_{i^*}.
4) Adjust the selected weight vector

$$w_{i^*}(n+1) = w_{i^*}(n) + \epsilon(n)[x(n) - w_{i^*}(n)] \tag{5}$$

where n is the training time and $0 < \epsilon(n) < 1$.

5) Repeat Steps 1) through 4) for all training vectors.

The fairness function $\mathfrak{F}(u_i)$ provides a simple way to control the behavior of the FSCL training procedure. For example, choosing $\mathfrak{F}(u_i) = 1$ reduces FSCL to the standard CL algorithm. The choice $\mathfrak{F}(u_i) = u_i$ is one we have used frequently and appears to provide a good compromise between minimizing distortion and ensuring uniform codeword usage. It is also possible to make $\mathfrak{F}(u_i)$ training iteration dependent, for example $\mathfrak{F}(u_i) = u_i^{\beta e^{-t/T}}$, where t is the training iteration number and T is a constant. This choice for $\mathfrak{F}(u_i)$ initially emphasizes uniformity of codeword usage, but emphasizes minimizing the distortion as training progresses.

In the next section, we demonstrate the application of the FSCL network to the vector quantization of speech and images. The results obtained from the FSCL net are compared to those generated by the LBG algorithm.

IV. Examples of VQ Codebook Design Using Neural Networks

A. Vector Quantization of Speech

We have used the KSFM and the FSCL training algorithms for the vector quantization of linear predictive coding (LPC) parameters of speech signals [26]. The LPC parameters used were the autocorrelation coefficients

Exhibit M Page 225

$\{R(0), \cdots, R(p)\}$ from short-time windows of the speech signal. During the voiced speech segments, a form of pitch-synchronous analysis was done. The LPC residual signal was used to locate the pitch periods [27]. During unvoiced segments, the autocorrelation coefficients were computed over a 20 ms time window, with successive windows being overlapped by 10 ms. The speech sampling rate was 10 kHz. We performed both speaker-dependent and speaker-independent coding experiments, and these are described below.

1) Speaker-Dependent VQ: In this experiment, we compared the performance of the LBG, KSFM, and FSCL algorithms in encoding the speech data from a single male speaker. The training data consisted of about 14 000 vectors, where each vector was the 10 autocorrelation coefficients from an analysis frame. These data were from the analysis of 24 sentences, with each sentence being about 3 s in duration. The Itakura-Saito distortion measure [26] was used during both training and encoding.

Fig. 4 shows the average distortion in encoding the training data set as a function of the codebook size for the LBG, KSFM, and FSCL algorithms. The fairness function for the FSCL algorithm was chosen to be $\mathfrak{F}(u_i) = u_i$. The neighborhood for the KSFM method was chosen to be an arbitrary rectangular grid; for the size 128 codebook, for example, it was a 16×8 grid. Fig. 4 shows that the performance of the FSCL and LBG algorithms are very close; the KSFM method, in general, shows a higher distortion for all codebook sizes.

We compare the codeword utilization for all three methods for the size 128 codebooks in Figs. 5-7. Also shown in these figures is the average Itakura-Saito distortion in encoding the training data set for each of the methods, and the codebook entropy. The codebook entropy is computed as

$$E = -\sum_{i=1}^{128} p_i \log_2(p_i) \tag{6}$$

where p_i is the relative frequency with which codeword i was used in encoding the data set. Note that in the ideal case, where all codewords are utilized equally often, the value of the entropy is 7. The closer this value is to 7, the more uniform is the codeword usage. The FSCL network not only yields the highest entropy but also the lowest Itakura-Saito distortion among all of the methods. Thus, for this realistic application, the FSCL method yields a codebook design that is comparable to the codebook which is produced with the LBG method.

2) Speaker-Independent VQ: Only the FSCL algorithm was used in the speaker-independent speech coding experiments. The training data consisted of six sentences from each of six speakers, three male and three female. Each sentence was roughly 2 s in duration. The LPC analysis procedure was similar to that for the speaker-independent experiments, except that 15 autocorrelation coefficients were used for each analysis frame. Also, the Euclidean distance between the autocorrelation coefficient

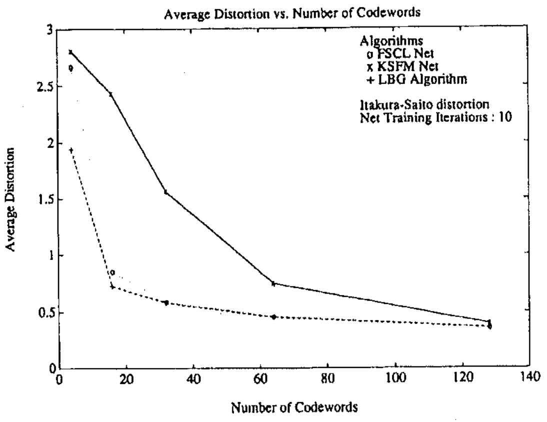


Fig. 4. Average distortion versus number of codewords, speaker-dependent VQ of speech.

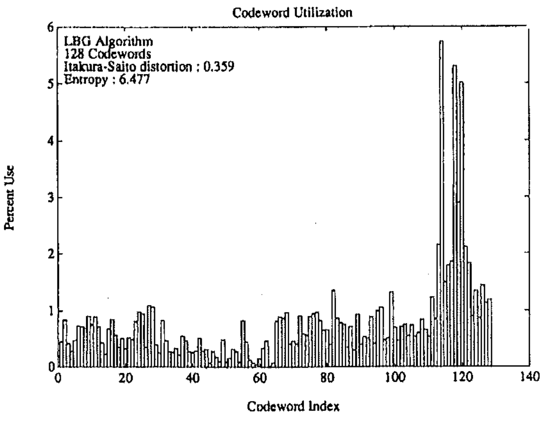


Fig. 5. Codeword utilization, LBG algorithm, speaker-dependent VQ of speech.

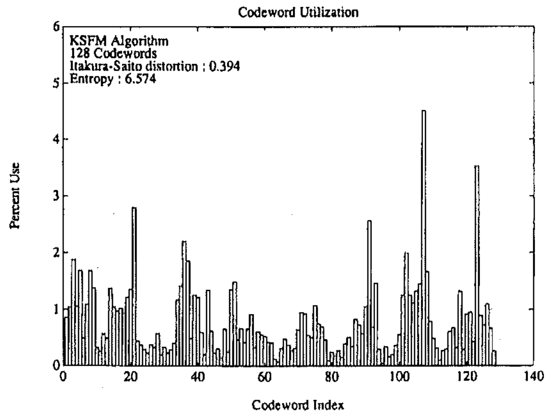


Fig. 6. Codeword utilization, Kohonen self-organizing feature map, speaker-dependent VQ.

Exhibit M Page 226

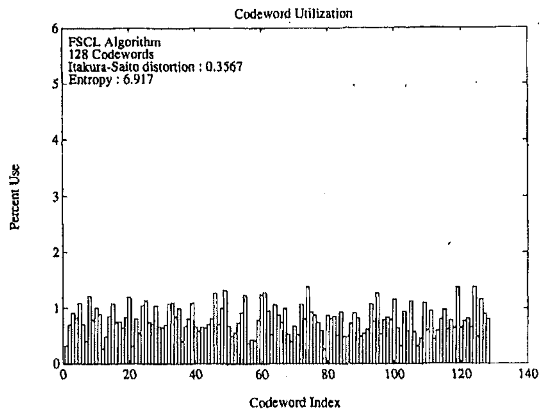


Fig. 7. Codeword utilization, frequency-sensitive competitive learning, speaker-dependent VQ.

vectors was used as the distortion measure. The size of the codebook was 128.

Table I shows the average Itakura–Saito distortion between the original and vector quantized autocorrelation parameters. Six sentences[1] that were used in training the VQ were encoded using the FSCL codebook. The average distortion in encoding these sentences is shown for the male and female speakers separately in the table. Six sentences from the same speakers that were not used in the training were also encoded, and the Itakura–Saito distortion for this case is also shown in the table. The results indicate that the performance is only slightly inferior as compared to the speaker-dependent case. Also, the performance is clearly better for the data used in training.

Fig. 8 shows the codeword utilization in encoding the training data set. The entropy for this case is 6.78, once again indicating uniform codeword usage.

Informal listening tests of speech synthesized using both the speaker-dependent and speaker-independent codebooks show that for the speaker-dependent case, the LPC synthesized and the vector-quantized and LPC synthesized speech sound almost identical; for the speaker-independent case, some additional distortion due to the VQ is evident.[2]

B. *Vector Quantization of Images*

In this section, we compare the LBG and FSCL algorithms applied to the task of designing codebooks for images. The techniques are applied to two images and the distortion and SNR are measured for various size codebooks. The results show that the neural network technique yields results that are very close to the optimal LBG design.

[1] One sentence from each of the speakers.

[2] Note that a codebook size of 128 is probably too small for the speaker-independent VQ.

TABLE I
ITAKURA–SAITO DISTORTION FOR SPEAKER-INDEPENDENT VQ OF SPEECH

	Included in Training	Not included in training
MALE	0.2169	0.3302
FEMALE	0.3826	0.6733
OVERALL	0.2997	0.5018

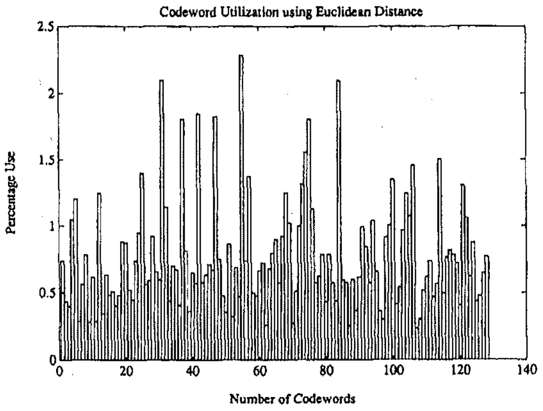


Fig. 8. Codeword utilization, frequency-sensitive competitive learning, speaker-independent VQ.

The images used in the experiments had a resolution of 500 × 482 pixels with 8 gray levels per pixel. A rectangular block of four pixels (2 × 2) was used as the vector to be quantized and codebook sizes were varied to provide performance comparisons between the two methods.

The convergence ration ϵ of the LBG algorithm was set to 0.01. The learning rate of the FSCL net was $\epsilon(t) = 0.01e^{-(u_i/10000)}$ and the distortion measure was modified by the inclusion of a "fairness" function $\mathfrak{F}(u_i)$, which depends on the number of times a particular unit wins the competition. $\mathfrak{F}(u_i)$ is nominally chosen to be u_i because of its simplicity and because reasonable results are experimentally obtained.

In Fig. 9, the original image is shown at 8 b/pixel. Fig. 10 shows the encoded images obtained when the bit rate is 1 b/pixel, with the LBG algorithm results in Fig. 10(a), and the FSCL algorithm results in Fig. 10(b). Finally, Fig. 11 shows the encoded images at a bit rate of 1.5 b/pixel, which is almost indistinguishable from the original image. Table II shows the mean-square-error and the signal-to-noise ratio as a function of codebook size for the FSCL and the LBG algorithms.

A second image containing significant edges was also used to study the encoding performance. Codebooks were designed with 16, 32, and 64 codewords. The "fairness" function was set to be $\mathfrak{F}(u_i) = u_i$, but the results obtained using the FSCL net are worse than those obtained using the LBG algorithm (Table III). We conjecture that the cause is that the image consists of nonuniform cluster sizes and choosing a fairness function of $\mathfrak{F}(u_i) = u_i$ results in undesirable uniformity in the use of the weight vectors.

Exhibit M Page 227

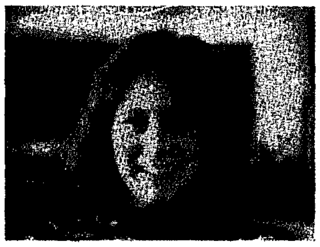
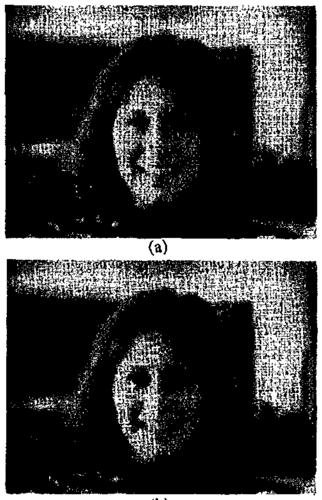


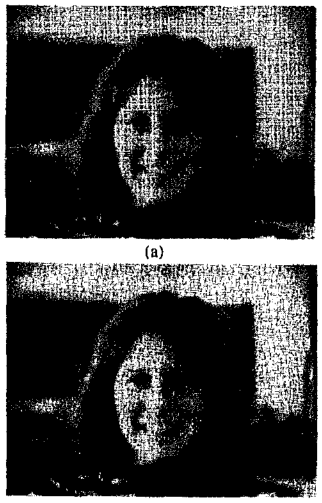
Fig. 9. Original image, 500 × 482 pixels, 8 b/pixel.



(a)

(b)

Fig. 10. Restored image using (a) LBG algorithm and (b) FSCL at 1 b/pixel.



(a)

(b)

Fig. 11. Restored image using (a) LBG algorithm and (b) FSCL at 1.5 b/pixel.

TABLE II
MEAN-SQUARE ERROR AND SIGNAL-TO-NOISE RATIO AS A FUNCTION OF CODEBOOK SIZE FOR THE LBG AND FSCL ALGORITHMS

Code-book Size	Bit Rate bits/pix	LBG Algorithm MSE	LBG Algorithm SNR (dB)	FSCL Net MSE	FSCL Net SNR (dB)
4	0.50	166.95	23.02	167.28	23.01
8	0.75	47.81	28.45	48.27	28.41
16	1.00	19.50	32.35	20.39	32.16
32	1.25	12.85	34.16	13.49	33.95
64	1.50	10.03	35.24	10.38	35.09

TABLE III
MEAN-SQUARE ERROR AS A FUNCTION OF CODEBOOK SIZE FOR FIXED AND DECREASING $\mathfrak{F}(u_i)$

Code-book Size	Bit Rate bits/pix	LBG $\epsilon = 0.01$ MSE	FSCL Net $\mathcal{F}(u_i) =$ u_i MSE	$u_i^{\beta e^{-t/T}}$ MSE
16	1.00	174.63	197.52	180.54
32	1.25	93.28	116.41	93.67
64	1.50	46.15	71.39	49.19

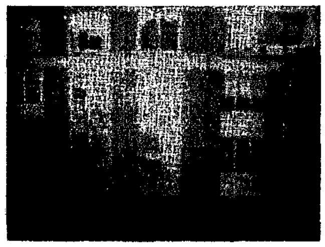


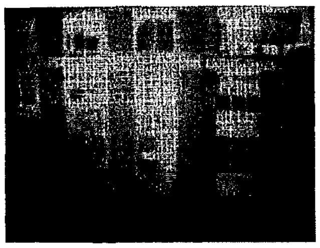
Fig. 12. Original image, 500 × 482 pixels, 8 b/pixel.

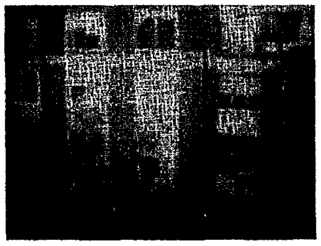
To determine if alternative fairness functions could improve the performance of the FSCL net, we repeated the experiment with a decreasing power function for u_i of the form $\mathfrak{F}(u_i) = u_i^{\beta e^{-t/T}}$, where t is the global count of training vectors presented to the net. This ensures mobility of the reference vectors at the beginning of the learning phase and as training proceeds the FSCL net gradually turns into competitive learning since $\lim_{t \to \infty} u_i^{\beta e^{-t/T}} = 1$. Consequently, as training progresses, the net eventually chooses the *winner* node with only the square error determining the error. In this case, β has been set to 1 and the value T has been set such that $\mathfrak{F}(u_i) = u_i^{e^{-1}}$ when the FSCL net has completed half of the learning process. The mean square error for this case also is shown in Table III.

Fig. 12 shows the original image. In Fig. 13 we show the restored image using the LBG algorithm [Fig. 13(a)] and the two versions of the FSCL net [Fig. 13(b), (c)].

We observe that the fairness function has a significant effect on the performance of the FSCL net. Rather than using a fixed power of β for the fairness function u_i^{β}, a decreasing power allows for better adaptation to the underlying distribution. We believe that this allows for a

Exhibit M Page 228


(a)


(b)

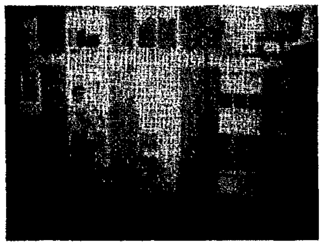

(c)

Fig. 13. Restored image using (a) LBG algorithm; (b) FSCL net with $\mathcal{F}(u_i) = u_i$; and (c) FSCL net with $\mathcal{F}(u_i) = u_i^{\beta e^{-t/T}}$, all at 1 b/pixel bit rate.

high mobility of the reference vectors during the initial stages of learning and then allows the FSCL net to turn into a competitive learning net. Further studies are underway to completely describe the effects of the fairness function.

V. Conclusions

This paper described the application of neural networks for vector quantization. The Kohonen self-organizing feature map and the frequency-sensitive competitive learning algorithms were discussed and shown to be useful for vector quantization codebook design. An important feature of the neural network approach to vector quantization is that the codebook design process is adaptive, and can consequently lead to adaptive VQ techniques.

We applied the neural network VQ methods for encoding speech and image data. The results of the algorithm were compared to the commonly used LBG design algorithm, and indicate that neural network VQ methods yield comparable results. We are presently investigating an adaptive VQ technique using neural networks.

Acknowledgment

The authors wish to thank T. P. Jung and R. Nallasivam for their assistance.

References

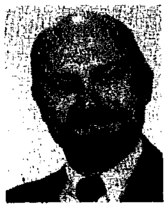
[1] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. COM-28, pp. 84-95, Jan. 1980.
[2] R. M. Gray, "Vector quantization," *IEEE ASSP Mag.*, vol. 1, no. 2, pp. 4-29, Apr. 1984.
[3] J. Makhoul, S. Roucos, and H. Gish, "Vector quantization in speech coding," *Proc. IEEE*, vol. 73, pp. 1551-1588, Nov. 1985.
[4] A. Buzo, A. H. Gray, Jr., R. M. Gray, and J. D. Markel, "Speech coding based upon vector quantization," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-28, no. 6, pp. 562-574, Oct. 1980.
[5] G. A. Davidson, P. R. Cappello, and A. Gersho, "Systolic architectures for vector quantization," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, pp. 1651-1665, Oct. 1988.
[6] R. R. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, vol. 4, no. 2, pp. 4-22, Apr. 1987.
[7] E. S. Grossberg, *Neural Networks and Natural Intelligence.* Cambridge, MA: M.I.T. Press, 1988.
[8] T. Kohonen, *Self-Organization and Associative Memory, 2nd Ed.* New York: Springer-Verlag, 1988.
[9] D. E. Rumelhart, J. L. McClelland *et al.*, *Parallel Distributed Processing.* Cambridge, MA: M.I.T. Press, 1986.
[10] S. Grossberg, "Competitive learning: From interactive activation to adaptive resonance," *Cognitive Sci.*, vol. 11, pp. 23-63, 1987.
[11] D. E. Melton, "Vector quantization of speech using neural networks," M.S. thesis, Ohio State Univ., Dec. 1988.
[12] N. M. Nasrabadi and Y. Feng, "Vector quantization of images based upon the Kohonen self-organizing feature maps," in *Proc. IEEE Int. Conf. Neural Networks*, 1988, pp. 1101-1108.
[13] J. Naylor and K. P. Li, "Analysis of a neural network algorithm for vector quantization of speech parameters," in *Proc. 1st Ann. INNS Meet.*, Boston, MA, 1988, p. 310.
[14] Y. Zhou, R. Chellappa, A. Vaid, and B. K. Jenkins, "Image restoration using a neural network," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, pp. 1141-1151, July 1988.
[15] F. H. Wu and K. Ganesan, "Comparative study of algorithms for VQ design using conventional and neural-net based approaches," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, 1989, pp. 751-754.
[16] S. C. Ahalt, A. K. Krishnamurthy, P. Chen, and D. Melton, "Vector quantization using frequency-sensitive competitive-learning neural networks," presented at the IEEE Int. Conf. Syst. Eng., Fairborn, OH, Aug. 24-26, 1989.
[17] T. Lee and A. M. Peterson, "Adaptive vector quantization using a self-development neural network," *IEEE J. Select. Areas Commun.*, this issue, pp. 1458-1471.
[18] J. Markel and A. Gray, *Linear Prediction of Speech.* New York: Springer-Verlag, 1976.
[19] S. C. Ahalt, A. K. Krishnamurthy, P. Chen, and D. E. Melton, "Competitive learning algorithms for vector quantization," *Neural Networks*, vol. 3, no. 3, pp. 277-290, 1990.
[20] T. Kohonen, *Self-Organization and Associative Memory.* New York: Springer-Verlag, 1984.
[21] R. Hecht-Nielsen, "Application of counterpropagation networks," *Neural Networks*, vol. 1, no. 2, pp. 131-141, 1988.
[22] S. Grossberg, "Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors," *Biological Cybern.*, vol. 23, pp. 121-134, 1976.
[23] J. H. Winters and C. Rose, "Minimum distance automata in parallel networks for optimum classification," *Neural Networks*, pp. 127-132, 1989.
[24] D. S. Bradburn, "Reducing transmission error effects using a self-organizing network," presented at the Int. Joint Conf. Neural Networks, Washington, DC, June 18-22, 1989.
[25] D. DeSieno, "Adding a conscience to competitive learning," in *Proc. IEEE Int. Conf. Neural Networks*, 1988, pp. 1117-1124.
[26] D. O'Shaughnessy, *Speech Communication: Human and Machine.* Reading, MA: Addison-Wesley, 1987.

Exhibit M Page 229

[27] Entropic Speech, Inc., *Entropic Signal Processing Systems*. Apr. 1989.

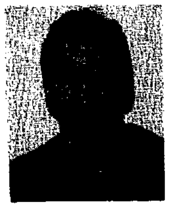
Ashok K. Krishnamurthy (S'82–M'83) received the B.Tech. degree from the Indian Institute of Technology, Madras, India, in 1979 and the M.S. and Ph.D. degrees from the University of Florida, Gainesville, in 1981 and 1983, respectively.

He has been an Assistant Professor in the Department of Electrical Engineering at The Ohio State University, Columbus, since 1986. His research interests are in speech synthesis and recognition and applications of neural networks.



Stanley C. Ahalt (S'77–M'79–S'79–M'80–S'82–M'85–S'85–M'86) received the B.S.E.E. and the M.S.E.E. degrees from Virginia Polytechnic Institute and State University, Blacksburg, in 1978 and 1980, and the Ph.D. degree from Clemson University, Clemson, SC, in 1986.

He was a Member of the Technical Staff at Bell Telephone Laboratories from 1980 to 1981, where he developed local area network hardware. He is currently an Assistant Professor with the Department of Electrical Engineering at The Ohio State University, Columbus. His research interests are in neural networks and parallel computing. He is the recipient of several research grants from Cray Research, under which he directed the development of the Neural Shell, a general purpose neural network simulation package.



Douglas E. Melton was born in Colorado Springs, CO, on May 11, 1965. He received the B. S. degree in electrical engineering from The Wichita State University, Wichita, KS, in 1987, and the M.S. degree in electrical engineering from The Ohio State University, Columbus, in 1988.

While at The Ohio State University, he was a Research Assistant and investigated the application of neural networks to speech coding. He also worked on the development of a speech intelligibility measurement system during an appointment as a Summer Fellow of the USAF Graduate Research Program in Dayton, OH. He is currently a Ph.D. student at The University of Wisconsin, Madison, in the Electroacoustics Laboratory. His interests are in the area of discrete-time recurrent neural networks, adaptive filtering, and active noise attenuation.

Prakoon Chen was born in May 1966. He received the B.S. and M.S. degrees in electrical engineering from The Ohio State University, Columbus, in 1987 and 1989, respectively. His M.S. thesis dealt with image vector quantization using neural network algorithms.

His interests are in vector quantization, neural network algorithms, and the mapping of these algorithms to parallel and vector computing machines. He is currently a Member of the Technical Staff at Oracle Corporation, Belmont, CA.

Mr. Chen is a member of Eta Kappa Nu.

Exhibit M Page 230