Vector Quantization using Artificial Neural Network Models [†]

Stanley C. Ahalt and James E. Fowler
Department of Electrical Engineering
Dreese Laboratory
The Ohio State University
Columbus, Ohio 43210

Abstract

This paper focuses on recent developments in the use of Artificial Neural Networks (ANNs) for Vector Quantization (VQ). A review of the fundamental ANN models used for VQ is presented, including Competitive Learning networks, Kohonen Self-Organizing Feature Maps, and Conscience Techniques including the FSCL algorithm. The paper also briefly reviews the use of VQ-based clustering techniques in classifiers, including Learning Vector Quantizers, Radial Basis Function Classifiers, and the ART architectures.

The paper then addresses some of the difficulties associated with the use of vector quantization in practical applications. In particular we focus on the use of VQ techniques for image data compression. While it has long been argued that one of the attractive features of ANNs is that they are readily adaptable to real-time hardware, this goal has been elusive. This paper discusses our successful efforts to construct a real-time Differential Vector Quantizer (DVQ) using ANN-inspired VLSI processors. This discussion concentrates on coding performance and entropy characteristics, and we also describe the DVQ architecture we have implemented.

I. Introduction

In many digital systems, even those in which substantial channel bandwidth is available, e.g., those employing fiber optic communication links, there is a pressing need to compress source signals. The goal is to transform a continuous-time signal into a representation that can be easily processed, transmitted, or stored. Vector quantization is one technique which encodes a discrete-time signal in order to both quantize and compress the data.

Typically, applications which employ VQ are those which require large amounts of storage or transmission bandwidth and can tolerate some loss of fidelity for the sake of compression. Vector quantization has been shown to be useful in compressing data that arises in a wide range of applications, including image processing [1, 2], speech processing [3], facsimile transmission [4], and weather satellites [5]. However, practical use of VQ techniques has been limited because of the prohibitive amount of computation associated with existing codebook design and encoding algorithms [6].

[†]Appears in *Proceedings of the International Workshop on Adaptive Methods and Emergent Techniques for Signal Processing and Communications* (D. Docampo and A. R. Figueras, eds.), (Bayona, Spain), pp. 42-61, June 1993.

This paper discusses the application of Artificial Neural Networks (ANNs) to VQ. Some of the attractive features of ANNs for VQ are that: 1) ANNs are based on biological processing systems that exhibit interesting, yet elusive computational properties, 2) ANNs are inherently parallel and permit highly parallel implementations, and 3) ANNs are adaptive, implying the possibility of adaptive vector quantizers. Our ongoing research focus is directed towards implementing real-time vector quantizers for image compression. The discussion here includes a description of a Competitive Learning (CL) training algorithm developed for efficient design of VQ codebooks. This algorithm yields near-optimal results and is computationally more efficient than other ANN vector quantizers which have appeared in the literature.

The paper is organized as follows. First a review of VQ and the fundamental ANN models used for VQ are presented, including Competitive Learning networks, Kohonen Self-Organizing Feature Maps (KSFM), and Conscience Techniques which include the Frequency-Sensitive Competitive Learning (FSCL) algorithm. Next, in order to demonstrate the fundamental utility of VQ, the paper discusses the use of VQ-based clustering techniques in classifiers, or pattern recognizers. We briefly describe Learning Vector Quantizers, Radial Basis Function Classifiers, and the ART architectures.

We then turn our attention to more practical considerations and focus on the task of image compression using VQ. The paper discusses our design of a real-time VQ encoder. Performance characteristics and coding efficiency are considered from a perspective of both source and channel coding. This paper concludes with a discussion of a real-time Differential Vector Quantizer (DVQ) system which we have recently constructed. The DVQ system implementation uses an ANN VLSI processor.

II. Vector Quantization

A Vector Quantizer (VQ) statistically encodes data vectors. Typically the data vectors are discrete samples drawn from one-dimensional or multi-dimensional continuous-time signals, e.g., speech waveforms or image rasters. The objective of VQ is to efficiently encode the signal into a digital representation that compactly represents the original signal while retaining the essential information contained in the original signal. Three possible goals [7] of such a conversion are to:

- reduce the bandwidth required to transmit a signal, or maximize the quality of a signal transmitted over a limited-bandwidth channel, or
- reduce the memory required to store a signal, or maximize the quality of a signal stored on a limited-memory device, or
- produce a signal representation that retains all necessary information while minimizing the processing required by subsequent (downstream) algorithms.

While a detailed discussion of vector quantization is beyond the scope of this paper, a brief discussion of the fundamental concepts is presented below. A recent book by Gersho and Gray [7] provides a complete treatment of the theoretical underpinnings of VQ, as well as a complete discussion of practical issues in traditional VQ. Additional detailed discussion can be found in [8], [2], and [9]. Lloyd, in [10], discusses an algorithm for quasi-optimal quantization of data, with extensions by Linde, Buzo, and Gray (LBG) [11] to arbitrary distortion measures and to vectors. The LBG algorithm is widely used to design the codebooks for VQ. Additionally, a discussion of the use of

vector quantizers in speech applications can be found in [3] and its use in image coding is described in [1].

A. Basic Vector Quantization Concepts

Vector quantization techniques capitalize on the underlying structure inherent in the data being quantized. The space of the vectors to be quantized is divided into regions and a reproduction vector is calculated for each region. Given any data vector to be quantized, the region in which the vector resides is calculated and is then represented by the reproduction vector for that region. Instead of transmitting or storing a given data vector, a symbol which indicates the appropriate reproduction vector is used. This results in considerable savings in transmission bandwidth, albeit at the expense of some distortion.

A simple example can be used to elaborate the difference between *vector* quantization and *scalar* quantization. Consider data samples drawn from some distribution, as shown in Fig. 1a. Suppose

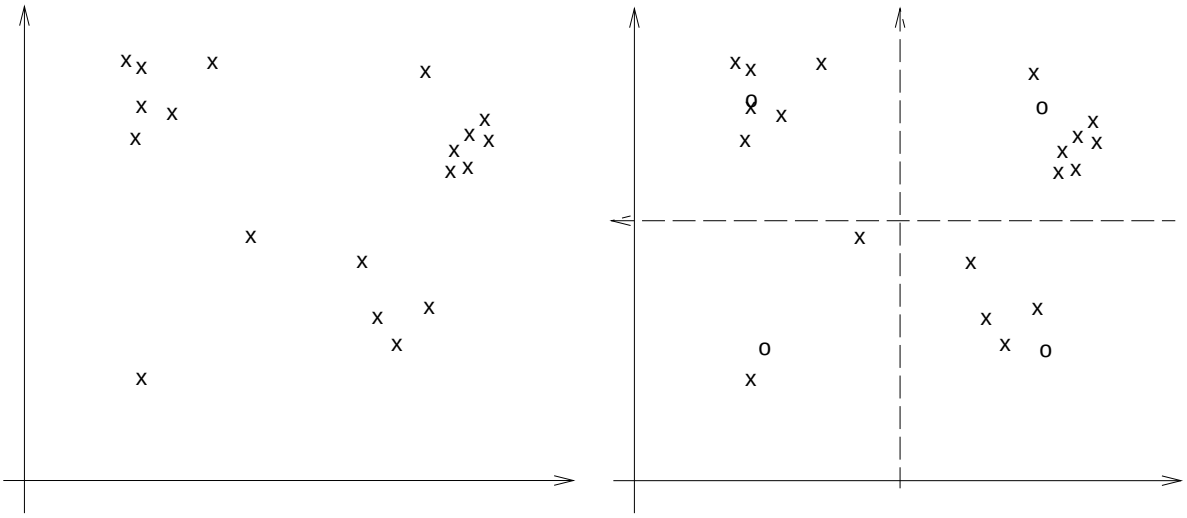

Figure 1: a) Data samples drawn from an arbitrary distribution (left) and b) one scalar quantization of the data (right). Input samples are denoted using x's, and the centroids of the quantization regions are denoted with o's.

this data is scalar quantized with a 2-bit index used to indicate the region from which the sample originated. One (rather naive) scalar quantization is shown in Fig. 1b, where the circles indicate the centroids of each of the regions. In effect, each of the data points in a region will be represented by the centroid of the rectangular region. The associated Mean Square Error (MSE) can be easily calculated from the average distance between each data point and its associated regional centroid. Of course, a more sophisticated scalar quantization can be realized by adjusting the centroids of each region based on the one-dimensional probability distribution function of each sample dimension, but the region will still be rectangular, and, in general, non-optimal (in the MSE sense).

Now consider the same set of samples, but consider the placement of the four centroids so as to minimize the MSE, as shown in Fig. 2a. This results in regions which are non-rectangular, in fact this procedure partitions the space into Voronoi regions, as shown in Fig. 2b. Thus, the

placement of the centroids to optimally minimize MSE must be done on a vector basis. This simple

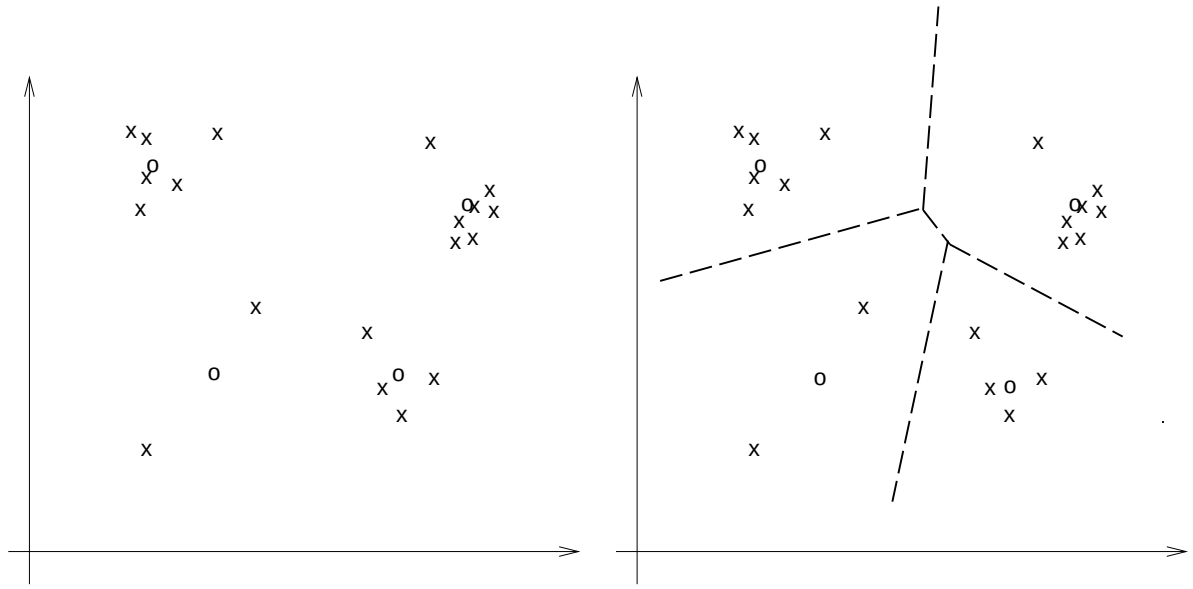

Figure 2: a) Data samples drawn from an arbitrary distribution with centroids placed to minimize MSE (left) and b) resulting Voronoi regions of the vector quantization of the data (right).

example demonstrates how, by considering the data as vectors and minimizing the distortion (MSE), considerably improved quantization can be realized. Note also that this process is essentially one of *clustering*. The centroids are commonly called *reproduction vectors*, *exemplars*, or *codewords*, while the collection of the reproduction vectors is called the *reproduction alphabet* or *codebook*.

More formally, vector quantization maps arbitrary data vectors to a binary representation or symbol. Thus, VQ is a mapping from a k-dimensional vector space to a finite set of symbols, $\mathbf{M}$. Associated with each symbol $m \in \mathbf{M}$ is a reproduction vector $\hat{\mathbf{x}}_m$. The encoding of the data vector $\mathbf{x}$ to the symbol m is a mapping, $VQ : \mathbf{x} = (x_1, x_2, \cdots, x_k) \rightarrow m$, where $m \in \mathbf{M}$ and the set $\mathbf{M}$ has size M. Assuming a noiseless transmission or storage channel, m is decoded as $\hat{\mathbf{x}}_m$, the reproduction vector associated with the symbol m which (ideally) minimizes some distortion metric. Since there are M elements in the set $\mathbf{M}$, there are M possible entries in the codebook. Once the codebook is constructed and, if necessary, transmitted to the receiver, the encoded symbol m acts as an index into the codebook. Thus, the *rate*, R, of the quantizer is $R = \log_2 M$ in bits per input vector. Since each input vector has k components, the number of bits required to encode each input vector component is R/k.

Because each data vector is represented as one of the codebook entries, the composition of the codebook determines the overall performance of the system. A number of different performance criteria can be used to determine an optimal codebook. For example, in speech and image transmission applications the usual objective is to minimize the overall distortion in the signal due to VQ. Thus the design criterion used to design an optimal codebook is the minimization of the average distortion in encoding vectors using the codebook.

Another possible criterion is to maximize the entropy of the codebook, i.e., to ensure that each of the codewords is used (approximately) equally frequently in encoding the data. The idea here is to

ensure that all the codewords are doing their "fair share" in representing the input data. As will be shown, this is a very useful criterion in developing ANN training algorithms for VQ design. In general, these two criteria are not equivalent; however, it can be shown that for the case when M is fixed and k is very large, the codebook which maximizes entropy also minimizes the expected distortion [3, 12].

For a specified performance criterion, VQ codebook-design techniques attempt to determine a codebook that is optimal with respect to this criterion. In general, this requires *a-priori* knowledge of the probability distribution of the input data. Typically, however, this distribution is not known[1], and the codebook is constructed through a *learning* or *training* process. During training, a set of data vectors that is representative of anticipated data is used to determine a quasi-optimal codebook.

During the training process, a distortion measure, $d(\mathbf{x}, \hat{\mathbf{x}})$ is used to determine which data points are to be considered as being in the same region. The distortion measure can be viewed as the cost of representing $\mathbf{x}$ as $\hat{\mathbf{x}}$. By determining which training data vectors lie in a particular region, the $k-$dimensional data space is partitioned into cells. All of the input vectors that fall into a particular cell are mapped to a single, common reproduction vector. If the cells are partitioned according to a minimum-distortion rule, then the partition is referred to as a Voronoi or Dirichlet partition.

A batch training process used to build a codebook proceeds as follows. Each of the data vectors is compared to the current codewords, and the corresponding distortion is calculated. The codeword that most closely matches the data vector, i.e., the reproduction vector which represents the input vector with minimum distortion, is selected and a codeword modification is calculated such that the modified codeword would reflect the inclusion of this new data vector in its partition. After the codeword modifications for all of the training vectors are calculated, each codeword is altered by an amount equal to the average of all of its calculated modifications.

As can be seen from this discussion, the training process is computationally expensive. Moreover, most of the algorithms currently used for VQ design, e.g., the LBG algorithm, are batch-mode algorithms [11] and need to have access to the entire training-data set during the training process. Because large training data sets are required to form an adequate representation of the the input vector space in certain applications (e.g., speaker independent speech coding), batch-mode training algorithms are problematic. Also, in many communication applications, changes in communication channels invalidate codebooks designed under different channel assumptions. Under these circumstances (both for handling large training-data sets and changing channel conditions), it is beneficial to work with adaptive VQ design methods, even if these are suboptimal in a theoretical sense. Adaptive VQ designs alter the codebook vectors with the arrival of each new training vector, and no "batching" of the training-data vectors occurs. One major advantage of formulating VQ as ANNs is that adaptive training algorithms that are used for ANNs can be applied to VQ. In the next section we discuss some of the ANN vector quantizers that have appeared in the literature.

III. Neural Networks for Vector Quantization

ANN techniques in VQ encoding and codebook design (training) have been demonstrated by number of researchers. For example, researchers [13] have used Kohonen Self-organizing Feature Maps [14]

[1]Even if the input distribution is known and relatively simple, it is not generally possible to solve for an optimal codebook, i.e., the problem does not permit a closed-form solution.

to construct VQ codebooks for speech applications, while others [15] used them to build VQ codebooks for image coding. Similarly, work with variable-region vector quantization of both speech and image is discussed in [16]. A brief review of the general framework of ANN vector quantization is given below.

A. Encoding

It is straightforward to formulate an ANN structure for the encoding of vectors. As before, let the vectors which are to be quantized be from a k–dimensional vector space, and let a distortion measure $d(\mathbf{x}, \mathbf{y})$ be defined in this space. Let the size of the codebook be M, and let the codewords be $\mathbf{c}_i$, $i = 1, \cdots, M$. Consider an ANN with M neural units, and make the i^{th} codeword, $\mathbf{c}_i$, the weight vector associated with neural unit i. Given any vector $\mathbf{x}$ that is to be encoded, $\mathbf{x}$ is fed in parallel to all the M neural units. Each of these units computes the distortion between the input vector and its weight vector, $d_i = d(\mathbf{x}, \mathbf{c}_i), \quad i = 1, \cdots, M$. The input vector is then encoded as the *index* i^* of the neural unit with the minimum distortion, $d_{i^*} = \min_j d_j$.

For a specific codebook, the "nearest-neighbor" encoding procedure described above is optimal [2]. Note that all computations, except for picking the "winning" neural unit and determining its index, are carried out in parallel. A number of ANN methods for picking the winner have been discussed in the literature. These include Grossberg's on-center, off-surround method [17], Lippmann et. al.'s MAXNET [18] and the Minimum Distance Automaton of Winters and Rose [19] . Also, as shown by Hecht-Nielsen [20], it is possible to directly compute the index number of the winning neural unit by adding an additional output layer to the ANN.

B. Training

We have shown in the previous section that ANNs can be used for encoding in vector quantizers. A more fundamental benefit of formulating vector quantization as an ANN task is that the large body of ANN training algorithms that have been developed can now be adapted to the problem of training vector quantizers. We discuss below three types of training algorithms: the Competitive Learning (CL) network, the Kohonen Self-organizing Feature Map (KSFM), and the Frequency-Sensitive Competitive Learning (FSCL) network.

1. The Competitive Learning Network

An adaptive version of the LBG algorithm for the training of vector quantizers can be realized with a Competitive Learning (CL) network. Assume that the ANN VQ is to be trained on a large set of training data. Further assume that the M neural units are initialized with the weight vectors $\mathbf{w}_i(o)$, $i = 1, \cdots, M$. These weights can be generated randomly or they can be the first M vectors of the training set.

Starting with these initial weight vectors, the training algorithm iterates a number of times through the training data, adjusting the weight vectors of the neural units after the presentation of each training vector. The algorithm used to adjust the weight vectors is based on competitive learning, which has been studied in depth by Grossberg [17, 21] and Kohonen [14, 22, 23] and by other researchers [20, 24, 25, 26].

The algorithm for updating the weight vectors is to first present the input vector $\mathbf{x}$ to all of the

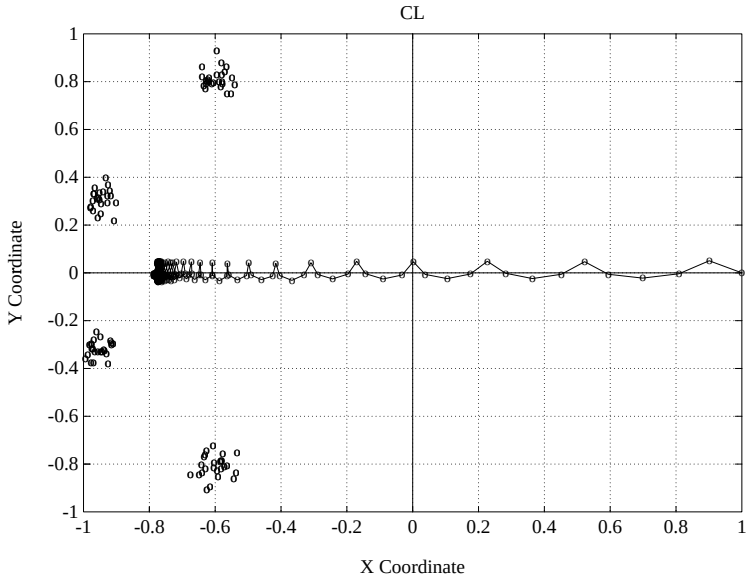

Figure 3: Movement of References Vectors, four data clusters, Competitive Learning.

neural units. Each unit then computes the distortion between its weight and the input vector. The unit with the smallest distortion is designated as the winner and its weight vector is adjusted towards the input vector.

Let $\mathbf{w}_i(n)$ be the weight vector of neural unit i before the input is presented. The output z_i of the unit is computed as follows.

$$z_i = \begin{cases} 1 & \text{if } d(\mathbf{x}, \mathbf{w}_i(n)) \leq d(\mathbf{x}, \mathbf{w}_j(n)),\ j = 1, \cdots, M \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The new weight vectors $\mathbf{w}_i(n+1)$ are computed as $\mathbf{w}_i(n+1) = \mathbf{w}_i(n) + \epsilon(\mathbf{x} - \mathbf{w}_i(n))z_i$.. In the above equation, the parameter ϵ is the learning rate, and is typically reduced monotonically as learning progresses. Note that this update rule is a form of Hebbian learning.

A significant problem with this training procedure is that it sometimes leads to neural units which are under-utilized. An excellent discussion of this problem can be found in [17] and subsequently in [24, 21]. Other researchers have used various methods to address this problem [20, 27]. The following example illustrates the problem of node under-utilization. The data for this example is from four clusters. An ANN with four units was trained using the above procedure. The initial weight vector of all the units was set to the same value of (1,0). As shown in Figure 3 only one of the weight vectors associated with the neural units is modified during training. The index of this single neural unit is consequently used for the quantization of data from all four clusters. Note that while the initial weight vectors for CL training are typically picked randomly; we have chosen the same value as an extreme example to illustrate the under-utilization problem of CL networks. Nevertheless, "regional" versions of this type of problem occur frequently in practical applications.

2. The Kohonen Self–Organizing Feature Map

Another important ANN algorithm that has been used for vector quantization is the Kohonen Self–organizing Feature Map (KSFM) [14, 13], initially conceived to illustrate the formation of topological feature maps in the brain. The KSFM and the Competitive Learning network are similar; however, in the KSFM algorithm each neural unit has an associated topological neighborhood of other neural units. During training the winning neural unit and the neural units in the neighborhood of the winner are updated. The size of the neighborhood is decreased as training progresses until each neighborhood has only one unit, i.e., the KSFM ANN becomes a CL net after sufficient training.

Let $\mathbf{w_i}(n)$ be the weight associated with the i^{th} neural unit, and let $\mathbf{x}$ be the input vector. Compute the distortion $d(\mathbf{x}, \mathbf{w_i}(n))$, $i = 1, \cdots, M$, and let the neural unit with the minimum distortion be i^*. Also, let $N(i^*)$ be the topological neighborhood associated with unit i^*. The weight update equations are:

$$\mathbf{w}_i(n+1) = \begin{cases} \mathbf{w}_i(n) + \epsilon(n)[\mathbf{x} - \mathbf{w}_i(n)], & i \in N(i^*) \\ \mathbf{w}_i(n), & \text{otherwise.} \end{cases} \tag{2}$$

Note that the gain sequence $\epsilon(n)$ slowly decreases with time to zero.

As can be seen from these equations, the KSFM structure involves more computation than the competitive learning network. At each step of the training process, the neighborhood, $N(i^*)$, of the winning node must be computed and all of the units in the neighborhood updated.

By the use of neighborhoods, the KSFM network overcomes the problem of under-utilized nodes discussed above. In Figure 4a we show the behavior of the KSFM for the 4 cluster problem previously discussed. As shown in Figure 4, all 4 neural units learn and move to the center of the 4 data clusters. All four weight vectors initially move together to the center of all four clusters of data. Then, once the neighborhood size is reduced to one, convergence to the data cluster centroids occurs. This example clearly illustrates the two phase learning mentioned by Kohonen [14].

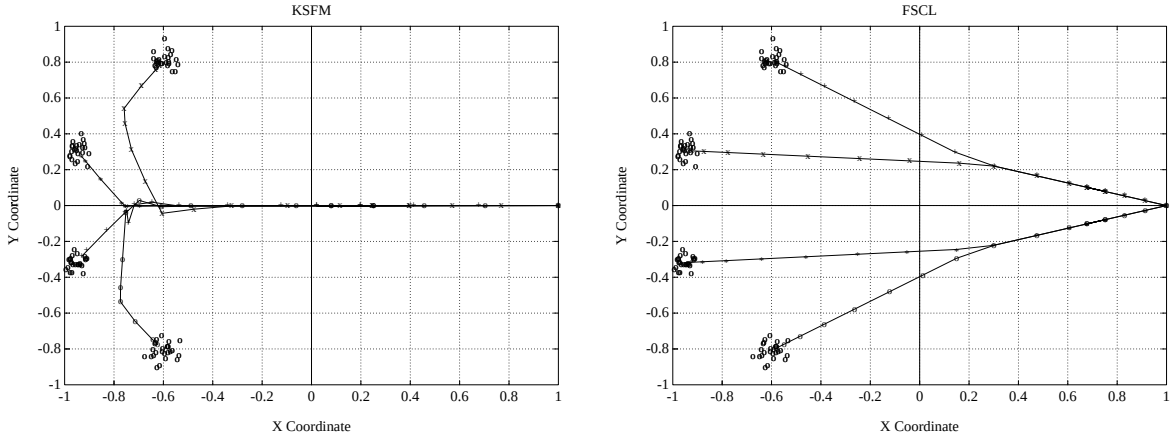

Figure 4: Trajectory of reference vectors for four cluster problem. a.) Movement of References Vectors, four data clusters, Kohonen Self-Organizing Feature Map. (left) b.) Movement of References Vectors, four data clusters, Frequency- Sensitive Competitive Learning (right).

One drawback of the KSFM network compared to the Competitive Learning network is the additional computation involved during training. This additional computation arises from both the calculation of the neighborhood of the winning unit, and from the updating of all members of the neighborhood. We describe below an alternative ANN that does not require the calculations associated with neighborhoods and solves the CL problem of under-utilized neural units.

3. Frequency-Sensitive Competitive Learning

One motivation of the Frequency-Sensitive Competitive Learning (FSCL) algorithm is to overcome the limitations of the simple Competitive Learning network while retaining its computational advantages. Since one of the main problems with the CL network is that some of the neural units may be under-utilized, the learning algorithm for the FSCL network keeps a count of how *frequently* each neural unit is the winner. This information is used to ensure that, during the course of the training process, all neural units are modified an approximately equal number of times. A similar approach was first described by Grossberg [17, 21], who suggested the use of a variable-threshold model to overcome the problem. Another similar approach, called the conscience method, has been suggested by DeSieno [27, 20] and subsequently used in slightly modified form in.

The FSCL method represents a method of applying the models introduced by Grossberg to the problem of Vector Quantization. In the FSCL network, each unit incorporates a count of the number of times it has been the winner. The distortion measure used to determine the winner is directly modified to include this count. Specifically, let $d(\mathbf{x}, \mathbf{w}_i(n))$ be the distortion to be minimized during the quantization process, and let $u_i(n)$ be the total number of times that neural unit i has been the winner during training. A modified distortion measure for the training process is defined as:

$$d^*(\mathbf{x}, \mathbf{w}_i) = d(\mathbf{x}, \mathbf{w}_i(n)) * \mathcal{F}(u_i(n)). \tag{3}$$

The "fairness function", $\mathcal{F}(.)$ is an increasing function of its argument during the early stages of training. The winning neural unit at each step of the training process is the unit with the minimum d^*. Note that if a given neural unit wins the competition frequently, its count and consequently d^* increase. This reduces the likelihood that this unit will be the winner, giving units with a lower count a chance to win the competition. However, no change is made to the learning rate.

In Figure 4b we illustrate the performance of the FSCL network on the same four cluster problem described previously. As can be seen from the figure, all four neural units are utilized and they move directly to the cluster centers. Thus, in the FSCL network, each neural unit dynamically adapts its chances of winning based on *both* the distortion and the number of times it has been modified. This property ensures that the neural units will be modified an (approximately) equal number of times.

In Table 1 we show codeword entropy and the average distortion for the various codebook design methods for a codebook size of 128. The training vectors for this example were the ten autocorrelation coefficients [28] obtained from short-time windows of a speech signal. Note that the FSCL method leads to the most uniform utilization of the codewords in the sense that each codeword is used an approximately equal number of times. The CL method utilizes one codeword 40% of the time and many of the neural units are never used to encode the data. The KSFM codebook shows greater uniformity in codebook utilization and results in an entropy of 6.574. This is consistent with the results reported by Naylor and Li [13] which show that the KSFM technique yields codebooks of more uniform utilization than those produced by covering followed by K-means clustering.

Table 1: Entropy comparison of various codebook design techniques, Itakura-Saito distortion.

Codebook Design Method	Entropy	Distortion
Competitive Learning	n/a	2.429
LBG	6.477	0.359
KSFM	6.574	0.394
FSCL	6.917	0.357

The FSCL network not only yields the highest entropy, 6.917, but also the lowest Itakura-Saito distortion among all of the methods.

IV. ANN VQ Classifiers

ANN techniques in VQ encoding and codebook design (training) have been demonstrated by number of researchers. In the next section of the paper we will discuss a specific example of how these VQ techniques can be used in one practical application, image compression. However, the VQ techniques discussed here are also applicable to a number of other tasks. In this section we briefly discuss the use of VQ techniques for classification tasks.

VQ is an example of *unsupervised* clustering. Unsupervised clustering techniques can be used to realize classifiers simply by associating *a posteriori* a class label with each reference vector. Then when an input vector is closest to a particular reference vector, the input vector is classified as belonging to the class of nearest reference vector. Multiple reference vectors can have the same class, so that non-circular class regions can be accommodated, at the cost of additional reference vectors. K-means classification is one example of this type of classifier.

In many situations, however, class information is available *a priori*. Typically this means that each training vector has an associated class label, and it is this data *pair* that is used during training. In most cases this information can be utilized so the objective becomes maximization of classification accuracy rather than, as in the case of of VQ clustering, distortion minimization.

Algorithms that attempt to optimally utilize available class information include Learning Vector Quantizers , Radial Basis Functions, and the ART architectures. A brief description of each of these techniques is given below.

4. Learning Vector Quantization

Learning Vector Quantization (LVQ) is a method developed by Kohonen [29] to "tune" the location of reference vectors in order to improve classification accuracy. The idea is to move reference vectors nearer to or farther away from decision surfaces in order to locate the decision surfaces as optimally as possible.

The most sophisticated technique is called LVQ3 [29], and can be described as follows. Let $\mathbf{w_i}(n)$ be the weight associated with the i^{th} neural unit, and let $\mathbf{x}$ be the input vector. Compute the distortion $d(\mathbf{x}, \mathbf{w_i}(n))$, $i = 1, \cdots, M$, and let the neural units with the minimum distortion and the next smallest distortion be i^* and i^{**} in either order. The weight update equations are:

$$\mathbf{w}_{i^*}(n+1) = \begin{cases} \mathbf{w}_{i^*}(n) - \epsilon(n)[\mathbf{x} - \mathbf{w}_{i^*}(n)], & \text{where } \mathbf{w}_{i^*} \text{ and } \mathbf{x} \text{ belong to different classes.} \end{cases} \quad (4)$$

$$\mathbf{w}_{i^{**}}(n+1) = \begin{cases} \mathbf{w}_{i^{**}}(n) - \epsilon(n)[\mathbf{x} + \mathbf{w}_{i^{**}}(n)], & \text{where } \mathbf{w}_{i^{**}} \text{ and } \mathbf{x} \text{ belong to the same class. Further, in both cases, } \mathbf{x} \text{ must fall into a symmetric window of non-zero width around the midplane separating } \mathbf{w}_{i^*} \text{ and } \mathbf{w}_{i^{**}}., \text{ or} \end{cases} \quad (5)$$

$$\mathbf{w}_i(n+1) = \begin{cases} \mathbf{w}_i(n) - \alpha\epsilon(n)[\mathbf{x} - \mathbf{w}_i(n)], & \text{where } i \in i^*, i^{**} \text{ and } \mathbf{x}, \mathbf{w}_{i^*} \text{ and } \mathbf{w}_{i^{**}} \text{ belong to the same class.} \end{cases} \quad (6)$$

(7)

A simple demonstration of this algorithm is shown in Fig. 5.

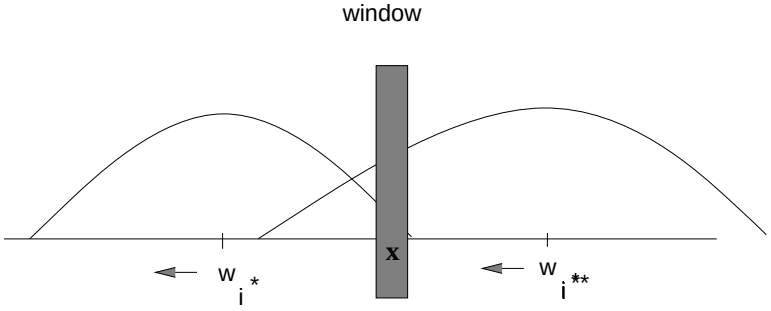

Figure 5: An example of the window and consequent reference vector movement in LVQ3.

This simple algorithm has been shown to yield results that, in practical tests, are superior to those of either the kNN algorithm or a parametric Bayesian classifier.

A. Radial Basis Function Classifiers

Radial Basis Function (RBF) classifiers are hybrid-architecture Artificial Neural Networks (ANNs) which consist of two layers[30, 31]. The first (hidden) layer (see Fig. 6) is an unsupervised layer and the second (output) layer is a supervised layer. The hidden layer uses locally receptive functions which are most strongly activated when the input feature vector is located at the mean of the function, i.e., a maximum response is obtained from one of the hidden units only when the input vector is equal to the mean of the particular hidden unit. The mean vectors are usually referred to as *prototype* vectors, and can be thought of as a catalog of common parametric models. These models will correspond directly to the generalized likelihood functions described above and are typically radially symmetric, normalized Gaussians.

In contrast with the techniques we have discussed previously, these activation functions do not compete, but are all simultaneously activated to varying degrees. If the input vector lies in the center of a unit's receptive field, then that unit will respond maximally and the remaining units will, at most, produce attenuated responses. On the other hand, if an input vector lies in two receptive fields, then both activation functions will respond (less that maximally), and the result is

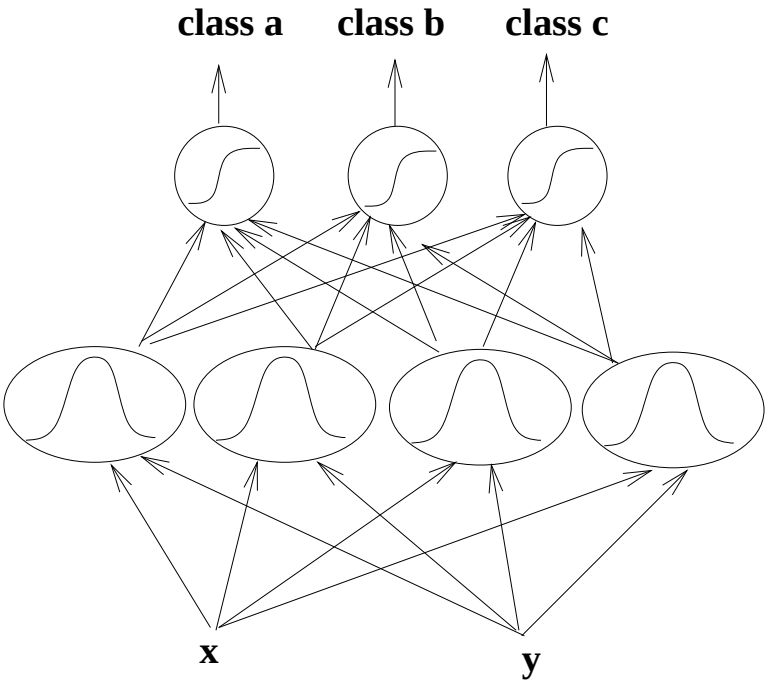

Figure 6: . A schematic overview of a Radial Basis Function Classifier.

passed on to the output layer. Since the responses are normalized, the pattern of activations across the receptive units can be interpreted as a measure of fit, or closeness, of the input vector to the reference vectors[32]. The metric which is used to determine the measure of closeness of each input vector to each receptive field prototype (center) can have a profound impact on the performance on the classifier. We have studied the effects of different metrics and have shown that the likelihood distributions and the *a priori* class probabilities of input data can suggest which metric to employ for a given classification domain [33].

The receptive field outputs are then passed on to the output layer for further processing. A supervised optimization procedure, e.g., backpropagation, is used to adjust the relative weighting of the contributions from each of the activation units. Obviously the system designer must use a suitably chosen output representation, a non-linear output activation function, and an appropriately crafted error function.

The relationship of RBF classifiers with traditional Bayesian classifiers is direct. For example, Richard and Lippmann [32] have recently shown that many neural-network classifiers provide outputs which estimate Bayesian *a posteriori* probabilities, allowing networks to be combined for higher level decision making. For RBF classifiers this means that the underlying models constituting the hidden layers must be either derived optimally from the *a priori* phenomenological knowledge or be derived directly from the training data. In the latter case the prototypes vectors can be readily determined using some form of clustering or *unsupervised* learning, e.g., FSCL [8].

B. Adaptive Resonance Theory

The Adaptive Resonance Theory (ART) classifiers are far too complex to adequately describe here. However, one very novel aspect of these architectures is that they effectively address a subtle

classification issue. In a typical *unsupervised* classifier (and in many supervised classifiers), every input vector is, of course, closest to one of the reference vectors and is thus classified as belonging to the class associated with the closest reference vector. The result is that input vectors which are quite dissimilar to *all* of the reference vectors are, nonetheless, classified as belonging to some class.

A more desirable property, in many applications, is to classify an atypical input vector as belonging to a class designated as "unknown" or "unusual". A classifier which exhibits this property effectively forms *closed* decision boundaries around each class, as shown in Fig. 7

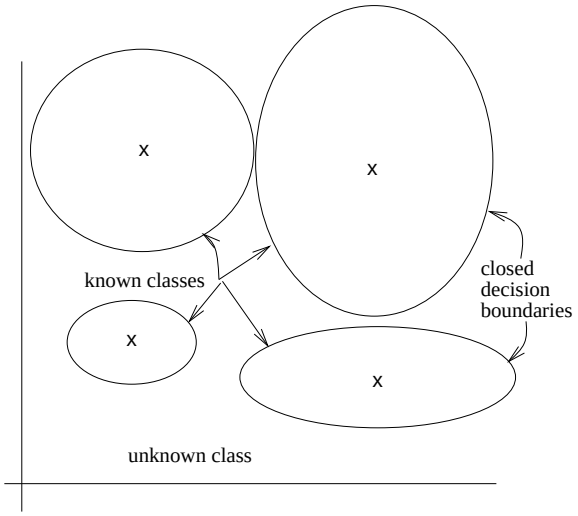

Figure 7: An example of closed decision boundaries.

The ART architectures realize closed decision boundaries while operating in an unsupervised mode. Closed decision boundaries, as well as the stable retention of long-term memory (LTM) - manifested as reference vectors [2] - are guaranteed by requiring that an input vector (Short-Term Memory) closely resemble a LTM vector before modifications to the reference vector are effected. A geometric explanation of how this property is realized is shown in Fig. 8.

A similarity vector, $\mathbf{R}$, is the sum of cP, a scaled version of the LTM pattern and $\mathbf{U_p}$, a normalized version of the STM. If $\mathbf{U_p}$ and cP point in the same direction, their sum, $\mathbf{R}$ has unit length and lies on the unit hypersphere. If $\mathbf{U_p}$ and cP point in different directions, their vector sum falls within the unit hypersphere. A "vigilance"parameter, ρ defines an inner hypersphere radius which is used to define how closely $\mathbf{U_p}$ and cP must lie in the same direction. If $\mathbf{R}$ is shorter than ρ and falls within the inner hypersphere then no learning occurs, i.e., no modification of $\mathbf{P}$ occurs.

This simple example demonstrates the mechanism by which an ART classifier 1) forms closed decision boundaries, and 2) solves the "stability-plasticity" problem. A *stable* classifier is one which preserves the values of reference vectors (LTM) for extended time even if the input distribution changes. A classifier exhibits *plasticity* if the classifier adapts to unique input vectors. One should note that other VQ-based techniques (simple VQ, KSFM, FSCL, LVQ, and RBFs) are operated in either a stable mode or a plastic (learning) mode. In contrast, an ART classifier is able to form

[2] In the following discussion, we will refer to the reference vectors as LTM memory vectors, and the input vectors as Short-Term Memory. This is an attempt to remain consistent with the terminology of Grossberg, but with a measure of imprecision. Strictly speaking, the LTM and STM vectors, as described by Grossberg are processed versions of the reference vectors and the input vectors. We take this liberty to in order to facilitate the discussion.

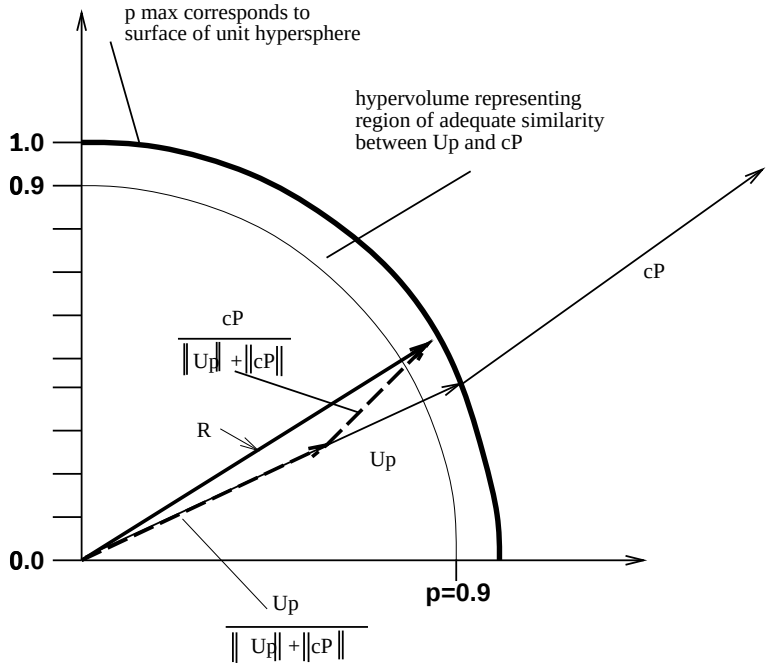

Figure 8: . The relationship between scaled LTM, cP, and normalized STM, $\mathbf{U_p}$.

stable LTM patterns while remaining plastic. This is accomplished by adding LTM nodes when no existing LTM pattern is found to satisfy the vigilance test.

This brief review of VQ-based classifiers has demonstrated how VQ techniques are applied to classification technology. In the following section, we return to our discussion of VQ and investigate the use of VQ for image compression.

V. A VQ Image Encoder Implementation

This section discusses an algorithm for compression of digital image data. The algorithm, known as Differential Vector Quantization (DVQ), combines differential pulse code modulation (DPCM) with vector quantization (VQ). The VQ codebooks are designed using Frequency-Sensitive Competitive Learning (FSCL) [8, 34]. Because, as discussed, FSCL attempts to maximize codebook entropy while minimizing distortion, additional entropy coding, which is commonly used to increase the compression of scalar DPCM schemes, is not needed. Therefore, the FSCL DVQ algorithm achieves compression rates similar to scalar DPCM methods while remaining robust in the presence of channel errors [35, 36]. The codebooks can also be structured so that variable-bit-rate encoding is supported without sacrificing insensitivity to channel errors.

Differential Vector Quantization (DVQ) combines the methods of vector quantization (VQ) and differential pulse code modulation (DPCM). DVQ replaces the scalar quantizer in the DPCM framework with a vector quantizer, and consequently has many of the compression advantages of

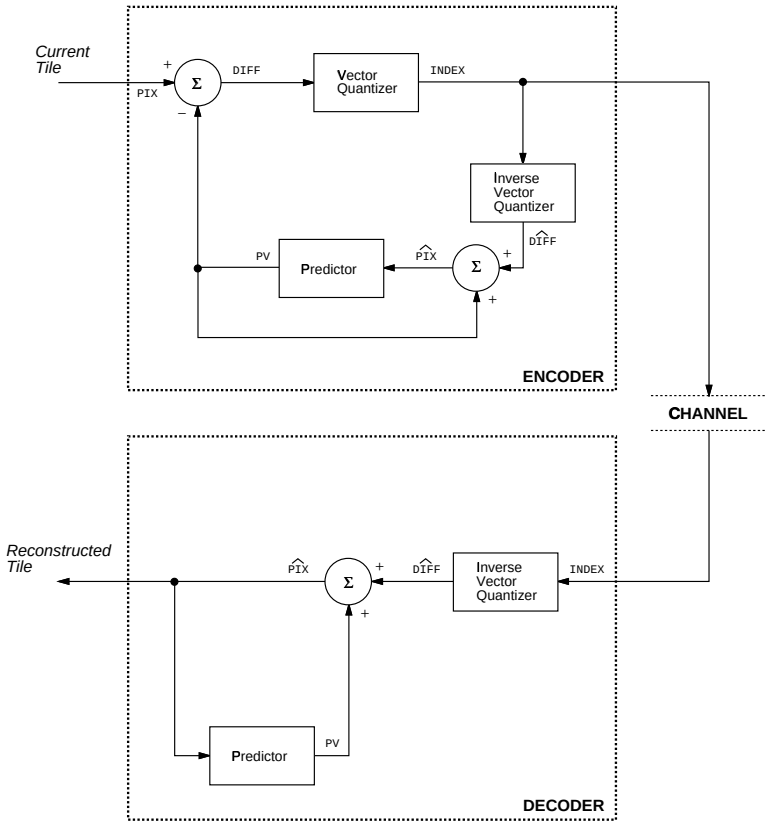

Figure 9: Differential vector quantizer algorithm block diagram

both VQ and DPCM. DVQ has been presented previously in [37], where it was called vector DPCM, and in [7], where it was called predictive vector quantization (PVQ). Other methods of combining vector quantization and prediction exist; see [38] and [7]. Figure 9 shows the general block diagram of the DVQ algorithm. A prototype of this encoder has been constructed using VLSI and MSI circuitry. The encoder runs in real time, operating at approximately 4 Gips.

DVQ has several advantages over both scalar DPCM and VQ. As mentioned above, the quantization of vectors yields better compression performance than that of scalars. Additionally, since the vector quantization is performed on difference values rather than on the image itself, the resulting image is less "blocky" [37]. Finally, the codebooks for DVQ tend to be more robust and more universally representative than codebooks designed for VQ because the difference tiles in a DVQ codebook are more generic than the image tiles in a VQ codebook [37].

There are many decisions to be considered in the design of a DVQ algorithm, such as tile size, distortion criterion, and method of prediction. These issues have been discussed in previous publications and so are omitted here. For more detail, refer to [35, 36].

The bit-rate of VQ coding may be varied by changing the number of codewords with which the encoding is done. That is, reducing the number of codewords present in the codebook results in an increase in compression ratio (a decrease in bit-rate). There are two simple methods of implementing such variable bit-rate coding within our DVQ algorithm. Each method allows coding to be done at a number of predefined bit-rates.

In the first method, the encoder stores a set of different codebooks, each of different size. Each

codebook has been trained independently on training images with the FSCL ANN algorithm. Consequently, we expect that each codebook has nearly maximal entropy for its size. The bit-rate is set by the choice of codebook used for coding. The encoder informs the decoder, which has an identical set of codebooks, of the codebook choice. We call this technique the *Trained* method of variable bit-rate DVQ coding.

In the second method, variable bit-rates are achieved by encoding using subsets of one fixed codebook. That is, the encoder and decoder have only one codebook, and tables of pointers define subsets within this codebook. The bit-rate is varied by selecting which subset of the codebook is used for coding. The encoder informs the decoder of the subset choice. We call this technique the *Extracted* method of variable bit-rate DVQ coding.

Several steps are involved in the incorporation of the *Extracted* method into the DVQ algorithm. Initially, codebooks of different sizes are trained with the FSCL ANN algorithm, as in the *Trained* method. The largest of these codebooks (say, of size n) is chosen as the codebook that will be stored in the encoder and decoder; this codebook is the basis codebook. Then, subsets are identified within this basis codebook; this process is called *extracting* codebooks. To extract a codebook of size m, where $m < n$, we identify the m codewords in the basis codebook which are closest, by Euclidean distance, to the m codewords in the smaller codebook. Then, we create a table of pointers to these m codewords. Tables of pointers are created in this fashion for each of the smaller codebooks extracted from the basis codebook.

Both the *Trained* and *Extracted* methods can easily be incorporated into a hardware implementation of the DVQ algorithm. It should be noted that the *Trained* method requires more storage space because it stores multiple codebooks. Also, the *Extracted* method is slightly more computationally complex since it uses pointers; however, this indirection has minimal impact on encoding speed.

A comparison of the *Trained* and *Extracted* methods described above is shown in Fig. 10. To generate these figures, ten grayscale images (8 bits per pixel) were coded with the DVQ algorithm at varying bit-rates, using both the *Trained* and *Extracted* methods described above. The basis codebook contained 256 codewords. It can be seen from these results that the *Trained* method provides slightly better mean squared error (MSE) at the same bit rate than does the *Extracted* method. However, the visual quality of the two methods at the same bit-rate is virtually identical.

Figure 11 presents the results obtained when errors are present in the transmission channel. Again, the *Trained* and *Extracted* methods perform similarly both with respect to MSE and visual quality. We have used an artificially high error rate (1 error per 1000 transmitted bits) to generate these results. However, our DVQ algorithm is quite insensitive to these channel errors, regardless of compression rate. In Fig. 12 we show that the amount of additional distortion due to channel errors is reduced as the size of the codebook is reduced. This effect is due to the fact that the codebook sorting algorithm we employed is more effective on smaller codebooks.

VI. Conclusions

This paper has focused on the use of Artificial Neural Networks (ANNs) for Vector Quantization (VQ). A review of the fundamental ANN models used for VQ was presented, including Competitive Learning networks, Kohonen Self-Organizing Feature Maps, and the FSCL algorithm. To more fully

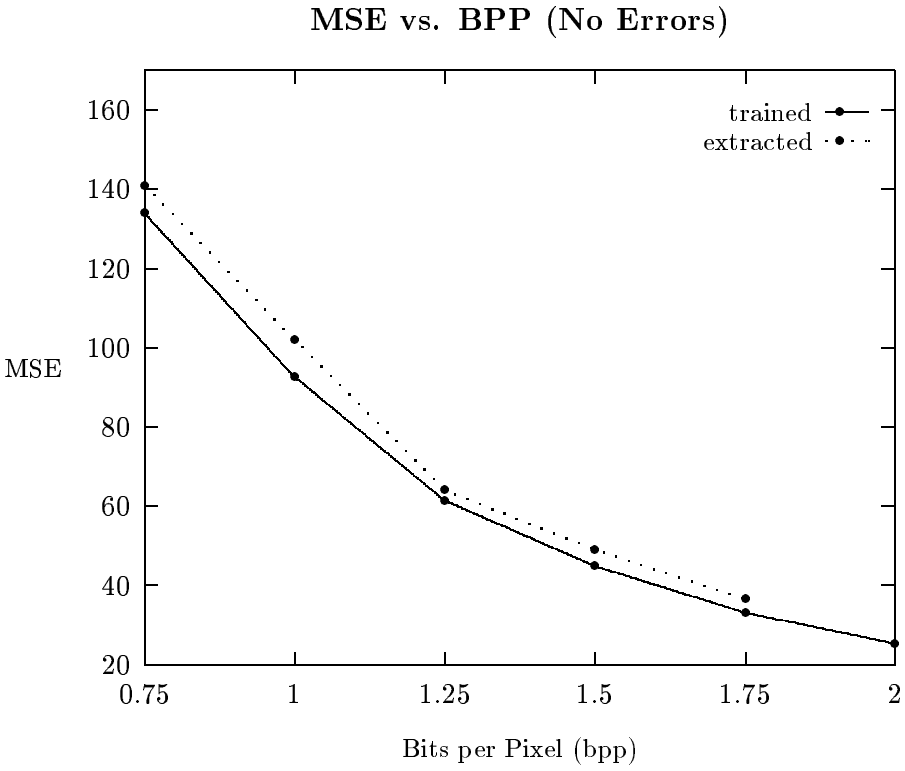

Figure 10: Rate-distortion curves for *Trained* and *Extracted* codebooks with no channel errors.

demonstrate the utility of these ideas, we then discussed the use of VQ techniques in constructing ANN-based classifiers. Finally, we discussed the use of ANN VQ in constructing real-time, robust image coders, focusing on the coding performance and entropy characteristics of these techniques, and the performance that can be realized when operating over noisy transmission channels.

VII. Acknowledgments

The interaction and assistance of Profs. Krishnamurthy and Bibyk has been essential to this research, and is deeply appreciated. Thanks also go to all the SPANN graduate students who have been instrumental in the work reported here, in particular the work of Ken Adkins in realizing the real-time DVQ encoder discussed here. Special thanks to NASA Lewis and Cray Research, Inc. who have supported much of this work.

References

[1] N. M. Nasrabadi and R. A. King, "Image Coding Using Vector Quantization: A Review," *IEEE Transactions on Communications*, vol. 36, pp. 957–971, August 1988.

[2] R. M. Gray, "Vector Quantization," *IEEE ASSP Magazine*, vol. 1, pp. 4–29, April 1984.

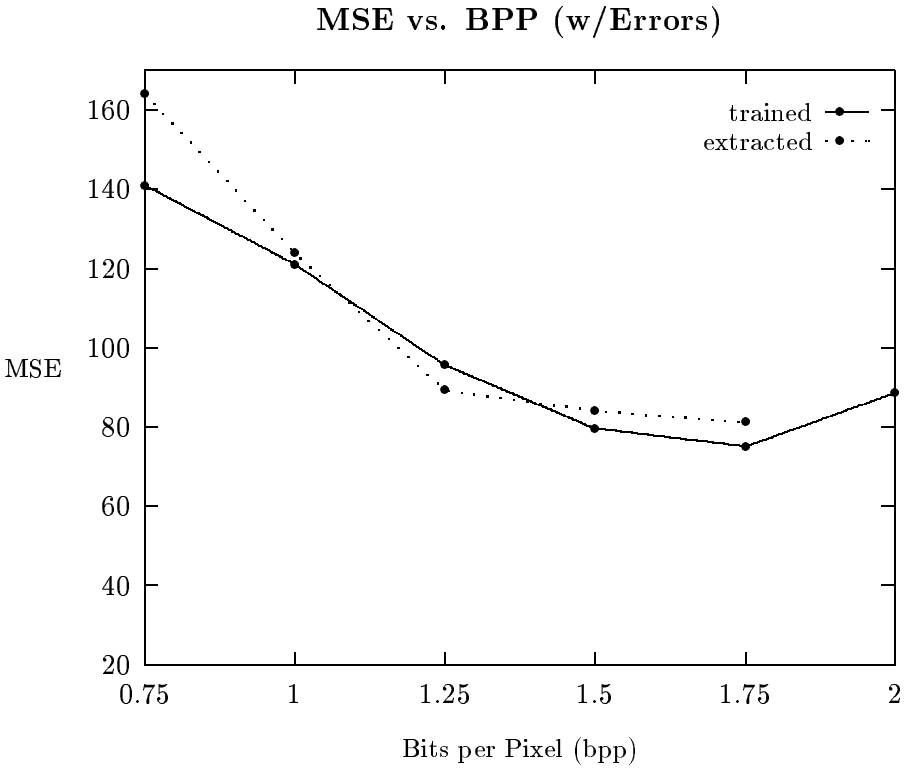

Figure 11: Rate-distortion curves for *Trained* and *Extracted* codebooks with channel errors.

[3] J. Makhoul, S. Roucos, and H. Gish, "Vector Quantization in Speech Coding," *Proceedings of the IEEE*, vol. 73, pp. 1551–1588, November 1985.

[4] A. N. Netravali and F. W. Mounts, "Ordering techniques for facsimile coding: A review," *Proceedings of the IEEE*, vol. 68, pp. 796–807, July 1980.

[5] C. Elachi, *Introduction to The Physics of Remote Sensing.* New York: Wiley Intersciences, 1987.

[6] G. Davidson, P. Capello, and A. Gersho, "Systolic Architechtures for Vector Quantization," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, pp. 1651–1664, October 1988.

[7] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression.* Kluwer international series in engineering and computer science, Norwell, MA: Kluwer Academic Publishers, 1992.

[8] S. C. Ahalt, A. K. Krishnamurthy, P. Chen, and D. E. Melton, "Competitive Learning Algorithms for Vector Quantization," *Neural Networks*, vol. 3, pp. 277–290, 1990.

[9] A. Gersho, "On the structure of vector quantizers," *IEEE Transactions on Information Theory*, vol. IT-28, pp. 157–166, March 1982.

[10] S. P. Lloyd, "Least-square quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, pp. 129–137, March 1982.

[11] Y. Linde, A. Buzo, and R. M. Gray, "An Algorithm for Vector Quantizer Design," *IEEE Transactions on Communications*, vol. COM-28, pp. 84–95, January 1980.

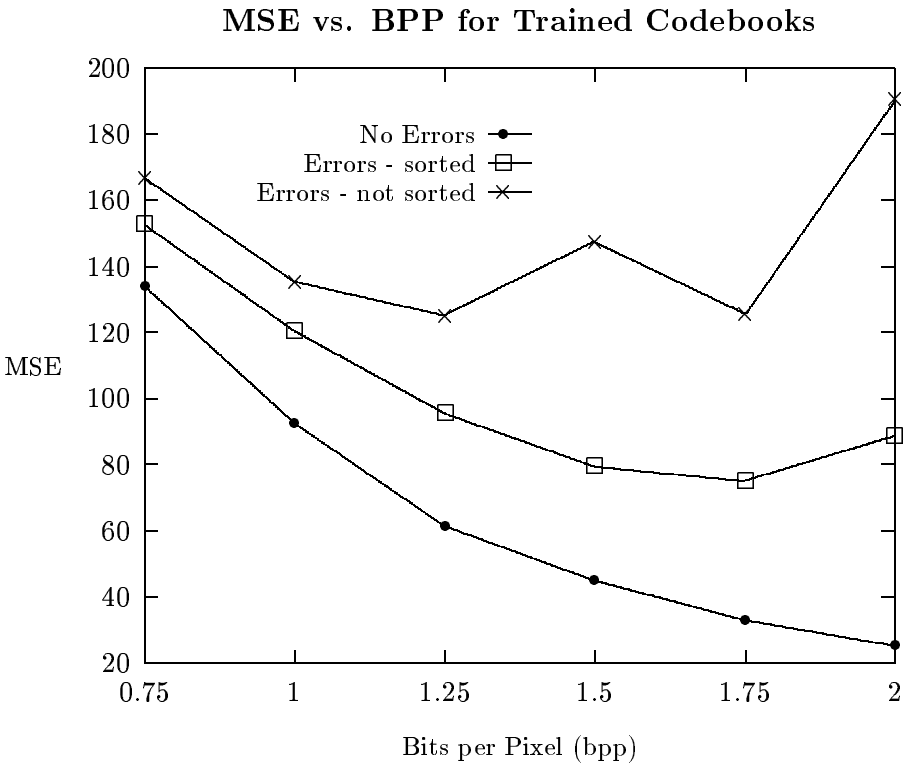

Figure 12: Rate-distortion curves for *Trained* and *Extracted* codebooks with channel errors.

[12] C. E. Shannon, "A mathematical theory of communication," in *Key Papers in The Development of Information Theory* (D. Slepian, ed.), pp. 5–18, New York: IEEE Press, 1948.

[13] J. Naylor and K. P. Li, "Analysis of a Neural Network Alogrithm for Vector Quantization of Speech Parameters," tech. rep., ITT Defense Communications Division, 10060 Carroll Canyon Rd., San Diego, CA 92131, 1988.

[14] T. Kohonen, *Self–Organization and Associative Memory, 2nd Ed.* Berlin: Springer–Verlag, 1988.

[15] N. M. Nasrabadi and Y. Feng, "Vector Quantization of Images Based on the Kohonen Self-Organizing Feature Maps," in *Proceedings of the IEEE International Conference on Neural Networks*, vol. I, (San Diego, CA), pp. 101–108, July 1988.

[16] Y. Matsuyama, "Vector Quantization with Optimized Grouping and Parallel Distributed Processing," tech. rep., Dept. of Information Science, Ibaraki University, 1990.

[17] S. Grossberg, "Adaptive Pattern Classification and Universal Recoding: I. Parallel Development and Coding of Neural Feature Detectors," *Biological Cybernetics*, vol. 23, pp. 121–134, 1976.

[18] R. P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, vol. 4, pp. 4–22, April 1987.

[19] J. H. Winters and C. Rose, "On Parallel Networks for Optimum Classification," tech. rep., AT&T Bell Laboratories, Holmdel, New Jersey 07733, 1988.

[20] R. Hecht-Nielsen, "Applications of Counterpropagation Networks," *Neural Networks*, vol. 1, no. 2, pp. 131–141, 1988.

[21] S. Grossberg, "Competitive Learning: From Interaction Activation to Adaptive Resonance," *Cognitive Science*, vol. 11, pp. 23–63, 1987.

[22] T. Kohonen, "An Introduction to Neural Computing," *Neural Networks*, vol. 1, no. 1, pp. 3–16, 1988.

[23] T. Kohonen, "Learning vector quantization," in *Abstracts of the First Annual INNS Meeting*, (Boston, MA), p. 303, Sept. 6 - 10, 1988.

[24] D. Rumelhart and D. Zipser, "Feature Discovery by Competitive Learning," *Cognitive Science*, vol. 9, pp. 75–112, 1985.

[25] D. E. Rumelhart, J. L. McClelland, *et al.*, *Parallel Distributed Processing.* Cambridge, Massachusetts: MIT Press, 1986.

[26] L. Steels, "Self-Organisation Through Selection," vol. II, pp. 55–62, 1988.

[27] D. DeSieno, "Adding a Conscience to Competitive Learning," in *Proceedings of the IEEE International Conference on Neural Networks*, vol. I, pp. 117–124, July 1988.

[28] D. O'Shaughnessy, *Speech Communication: Human and Machine.* Reading, Mass.: Addison-Wesley, 1987.

[29] T. Kohonen, "The Self-Organizing Map," *Proceedings of the IEEE*, vol. 78, pp. 1464–1480, September 1990.

[30] J. Moody and C. Darken, "Fast Learning in Networks Of Locally Tuned Processing Units," *Neural Computation*, vol. 1, pp. 281–294, 1989.

[31] T. Poggio and F. Girosi, "Networks for Approximation and Learning," *Proceedings of the IEEE*, vol. 78, pp. 1481–1497, September 1990.

[32] M. D. Richard and R. P. Lippmann, "Neural Network Classifiers Estimate Bayesian *a posteriori* Probabilities," *Neural Computation*, pp. 461–483, 1991.

[33] D. Hermann and S. A. R. Mitchell, "Clustering and Compression of High-Dimensional Sensor Data," in *SPIE International Symposium on Optical Engineering: Signal Processing, Sensor Fusion, and Target Recognition II*, pp. 286–297, April 1993.

[34] A. K. Krishnamurthy, S. C. Ahalt, D. Melton, and P. Chen, "Neural Networks for Vector Quantization of Speech and Images," *IEEE Journal on Selected Areas in Communications*, vol. 8, pp. 1449–1457, October 1990.

[35] M. R. Carbonara, J. E. Fowler, and S. C. Ahalt, "Compression of Digital Video Data Using Artificial Neural Network Differential Vector Quantization," in *Applications of Artificial Neural Networks III* (S. K. Rogers, ed.), pp. 422–433, Proc. SPIE 1709, 1992.

[36] A. K. Krishnamurthy, S. B. Bibyk, and S. C. Ahalt, "Video Data Compression Using Artificial Neural Network Differential Vector Quantization," in *Proceedings of the Second NASA Space Communications Technology Conference*, (Cleveland, OH), pp. 95–101, November 1991.

[37] C. W. Rutledge, "Vector DPCM: Vector Predictive Coding of Color Images," in *Proceedings of the IEEE Global Telecommunications Conference*, pp. 1158–1164, September 1986.

[38] H.-M. Hang and J. W. Woods, "Predictive Vector Quantization of Images," *IEEE Transactions on Communications*, vol. COM-33, pp. 1208–1219, November 1985.