Video Data Compression Using Artificial Neural Network Differential Vector Quantization

Ashok K. Krishnamurthy Steven B. Bibyk Stanley C. Ahalt

Department of Electrical Engineering
The Ohio State University
Columbus, Ohio 43210

Abstract

An artificial neural network vector quantizer is developed for use in data compression applications such as Digital Video. Differential Vector Quantization is used to preserve edge features, and a new adaptive algorithm, known as Frequency-Sensitive Competitive Learning, is used to develop the vector quantizer codebook. To develop real time performance, a custom VLSI ASIC is being developed to realize the associative memory functions needed in the vector quantization algorithm. By using vector quantization, the need for Huffman coding can be eliminated, resulting in superior performance against channel bit errors than methods that use variable length codes.

1 Introduction

Effective data compression algorithms are needed to reduce transmission bandwidth and storage space. In particular, there is a great deal of interest in the low bit rate coding of images. In this paper, we discuss the compression of digital video image data, which has become a central concern as HDTV standards begin to develop. One compression technique, Vector Quantization (VQ) [1, 2], has emerged as a powerful technique that can provide large reductions in bit rate while preserving essential signal characteristics. In this paper we show that error-insensitive VQ encoders can be constructed by employing *entropy* based VQ codebooks.

The purpose of this paper is to describe the use and implementation of an Artificial Neural Network (ANN) Vector Quantizer. More specifically, we discuss the design of a real-time, edge-preserving Differential Vector Quantizer (DVQ) architecture. We discuss the use of an ANN algorithm to design VQ codebooks, and we anticipate that the use of the same ANN algorithm can be employed in *adaptive* DVQ coders. The particular ANN algorithm we use is called Frequency Sensitive Competitive Learning (FSCL). This algorithm has been described in depth in previous publications [3, 4], so only a brief presentation is given here.

A locally-optimal vector quantization algorithm, proposed by Linde, Buzo, and Gray (LBG) [5], has been extensively employed in encoding both speech and images. However, studies have shown that, in many cases, the computational complexity of this algorithm restricts its use in real-time applications [1, 6]. The use of ANNs to perform vector quantization has been proposed to overcome these limitations.

The use of ANNs for vector quantization has a number of significant advantages. First, ANNs are highly-parallel architectures and thus offer the potential for real-time VQ. Second, the large body of training techniques for ANNs can be adapted to yield new, and possibly better, algorithms for VQ codebook design. Third, in contrast to the batch training mode of algorithms based on the LBG algorithm [7], most ANN training algorithms are adaptive; thus, ANN based VQ design algorithms can be used to build *adaptive* vector quantizers [8]. This is crucial in applications where the source statistics are changing over time.

This paper is organized as follows. First, we briefly describe basic Vector Quantization techniques and discuss ANN VQ techniques. We then describe the FSCL algorithm and show how the FSCL algorithm attempts to build a maximum-entropy codebook. Then, in Section 3, we describe how a VQ encoder can be viewed as an Associative Memory (AM) and discuss issues related to the design and implementation of an AM. This is followed by a short discussion on the Differential Vector Quantization architecture which is used to minimize edge distortion. We then present our experimental results in Section 5 where a FSCL codebook is used in a DVQ architecture to compress digital images.

2 Vector Quantization and the FSCL Artificial Neural Network

2.1 Basic Vector Quantization Concepts

Vector quantization capitalizes on the underlying structure of the data being quantized. The space of the vectors to be quantized is divided into a number of regions of arbitrary volume and a reproduction vector is calculated for each region. Given any data vector to be quantized, the region in which it lies is determined and the data vector 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 can result in considerable savings in transmission bandwidth, albeit at the expense of some distortion.

More formally, vector quantization maps arbitrary data vectors to a binary representation or symbol. Thus, the VQ mapping is 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. The collection of all possible reproduction vectors is called the *reproduction alphabet* or more commonly the *codebook*. 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 $R = \log_2 M$ 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.

Since each data vector must be ultimately 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 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 equally frequently in encoding the data. This is a very useful criterion in developing ANN training algorithms for VQ design because maximum entropy codebooks can be employed without the use of Huffman codes, thus reducing encoder sensitivity to channel errors. Finally, an alternative criterion is to use a distortion measure that incorporates expected responses of the human visual system to differences in intensity values and motion.

Given a performance criterion, the VQ codebook design process involves the determination of a codebook that is optimal with respect to this criterion. This normally requires knowledge of the probability distribution of the input data. Typically, however, this distribution is not known, and the codebook is constructed through a process called *training*. During training, a set of data vectors that is representative of the data that will be encountered in practice is used to determine an optimal codebook.

During the training process, a distortion measure, $d(\mathbf{x}, \hat{\mathbf{x}})$ is typically 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 the same 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.

2.2 Motivations for the use of ANN VQs

Unfortunately, the VQ training and encoding processes are computationally expensive. Moreover, most of the algorithms currently used for VQ design are *batch mode* algorithms [5], and need access the entire training data set during the training process. Using ANN adaptive techniques, it is possible to realize an adaptive VQ coder in which codewords are modified based on the arrival of each new training vector.

2.3 The FSCL Algorithm

The Frequency - Sensitive Competitive Learning (FSCL) algorithm is an unsupervised ANN consisting of two layers. The input layer nodes transmit the input vector elements to each of the nodes in the output layer. In the output layer, known as the *winner-take-all layer*, each node receives inputs from all of the input nodes. The weighted interconnections between these two layers are considered the *exemplar*, or *weight* vectors and are used for selecting the *winner node*. The winning node is selected on the basis of a modified distortion measure for each of the output layer nodes. The FSCL codebook design algorithm used in a training phase is discussed below.

One of the motivations for the Frequency-Sensitive Competitive Learning (FSCL) network is to overcome the limitations of simple competitive learning network while retaining its computational advantages. One of the main problems with CL networks 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 training process, all neural units are modified an approximately equal number of times. This yields a codebook that, on average, utilizes all of codewords equally. Consequently, the use of variable-length Huffman codes is unnecessary because no additional compression will be achieved through their use.

To solve the under-utilization problem and obtain an equiprobable codebook, the FSCL Algorithm uses a fairness function, $\mathcal{F}(u_i)$, which is a function of the local *update counter*, u_i, and is chosen to ensure the utilization of all the nodes in the winner-take-all layer. The motivation and use of fairness function has been discussed in previous papers [3, 4].

Finally, if the codewords are indexed or labeled such that codewords which are close in Hamming distance are also close in the chosen distortion criteria (e.g., absolute distance) then the resulting encoding architecture will be relatively insensitive to transmission errors. This is because random bit errors in the channel will result in reproduction vectors which are "close" in the distortion criteria chosen by the codebook designer.

2.3.1 The FSCL Training Phase

In the *training phase*, the exemplar vectors in the winner-take-all layer are adjusted adaptively to statistically reflect the distribution of the training vectors. A training vector is applied to the input layer and compared to all of the exemplar vectors in the winner-take-all layer. Upon the completion of the comparisons, one node in the winner-take-all layer is selected to be the *winner* and the rest of the nodes are inhibited. The selection of the winner node depends on the product of the fairness function, $\mathcal{F}(u_i)$, and the distortion measure. The distortion measure of each input training vector is calculated with respect to the exemplar vector. Common approaches for measuring the distortion of the input and exemplar vectors are dot product, Euclidean distance, and absolute distance.

The exemplar vector of the winning node is adjusted, by an amount specified by the learning rate, so as to more closely represent the input vector. Finally, the winner node increments its update-counter, u_i, and then the next training vector is presented to the network. Each node in the winner-take-all layer has a private update-counter, and the counters are used to influence the selection of the winner nodes. As a result, infrequently used exemplar vectors are adjusted, even if they had a larger distortion measure than other exemplar vectors [9].

When the learning phase is completed, all of the weight vectors in the winner-take-all layer are adapted, and the FSCL-derived codebook can be used in the encoder. As previously noted, the use of the modified distortion measure insures that the codewords are updated approximately the same number of times, thus maximizing the entropy of the codebook.

2.3.2 The Encoding and Decoding Phase

After the codebook has been constructed, input data vectors are coded by comparing each data vector with each of the codewords and then transmitting (or storing) the index of the (winning) codeword which yields the minimum distortion. Finding the minimum distortion codeword is a time-consuming task, but this operation is inherently parallel. The parallel hardware we have developed for the encoding is discussed in Section 3, and uses winner-take-all circuits in an Associative Memory encoder. Note that the winner-take-all circuits are used during training and encoding. However, only during training does adaptation of the codewords occur.

To decode the datum the receiver uses the received index to access a copy of the codebook to determine the reproduction vector used to represent the original data. This is a simple look-up process, and can be done without special hardware support in conventional RAM.

3 Vector Quantization as Associative Memory

VQ can be thought of as the process taking an input vector and matching it to the closest vector out of a set of vectors. Once the closest match is found, the index of the match is transmitted to the receiver. This process can also be viewed as an Associative Memory (AM), where the index is associated with a particular codeword.

As an example of an associative memory which uses Hamming distance and has very fast matching capabilities, consider the following design. The AM cell, shown in Figure 1, is a simple variant of the standard static RAM cell.

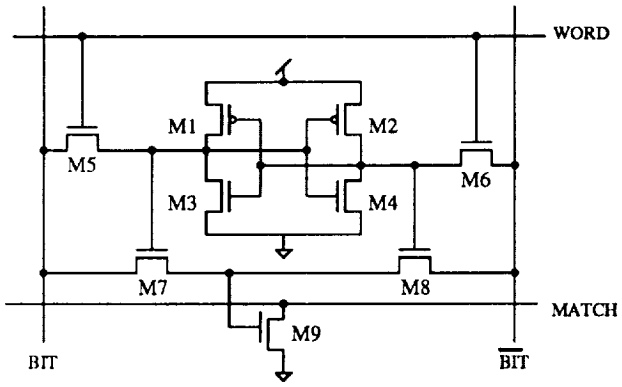

Figure 1: An Associative Memory cell. The transistors M7, M8 and M9 are the only additions to a standard static RAM. Their function is to draw the MATCH line low when the value on the BIT and $\overline{BIT}$ lines match the value stored in the static RAM core.

When in a matching mode, i.e. any time the WORD line is not asserted, if the value on the BIT line matches that stored by transistors M1 and M3, the transistor M7 becomes active and M9 is turned on and the MATCH line is drawn down. Similarly, if the $\overline{BIT}$ line matches the value stored in transistors M2 and M4, the transistor M8 becomes active and again M9 turns on.

The AM cells are arranged in an array such that the words to be matched all share the same MATCH line, as shown in Figure 2.

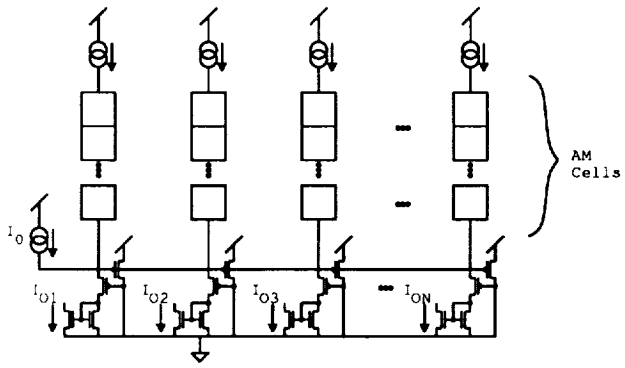

Figure 2: An array of associative memory cells, with their corresponding closest match circuitry. The circuitry which determines the closest match is a winner–take–all network which grows linearly with the number of words to match.

At the top of each column of AM cells is a current supply, which drives the MATCH line. In match mode as the value to be matched is shown to the column via the BIT and $\overline{BIT}$ lines, the AM cells begin to sink current if the value stored in the AM matches that presented on the BIT and $\overline{BIT}$ lines. The circuitry at the bottom then determines which current coming in is the highest. The only I_{OUT} with any output current is the I_{OUT} for which the corresponding MATCH line that is the largest. This I_{OUT} is then used to gate the value of the given AM column to the output buffers.

4 Differential Vector Quantization

Differential Pulse Code Modulation (DPCM) can be used to perform quantization of image data in a CODEC (COder–DECoder) architecture. One example can be found in [10] which is based on DPCM but also utilizes a nonuniform quantizer and multilevel Huffman coding to reduce the data rate substantially below that achievable with straight DPCM. As a result of variable length coding, the compression ratio is different for different images depending on the statistics of that particular image. Furthermore, because of the different codeword lengths it is very difficult to detect and compensate for transmission errors even if an end-of-line reset is used to re-sychronize the encoder and the decoder.

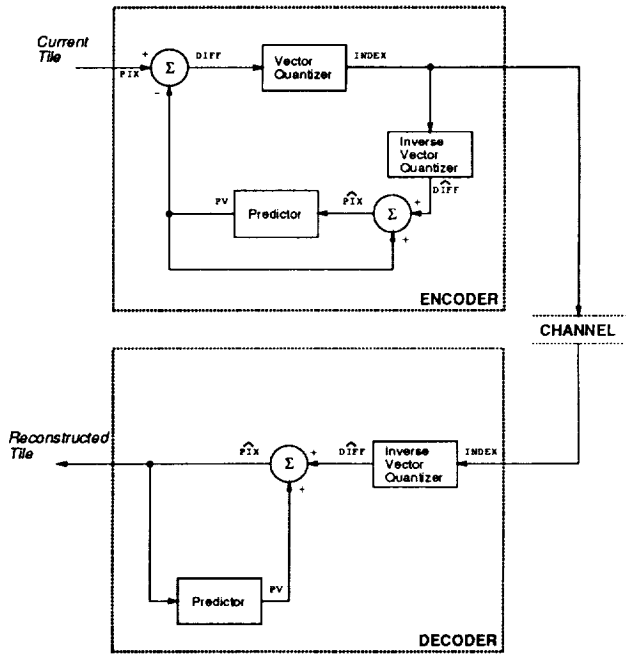

Figure 3: Block Diagram for Differential Vector Quantizer

Differential Vector Quantization (DVQ) (see Figure 3) incorporates the desirable qualities of both Differential Pulse Code Modulation (DPCM) and

VQ [11]. In DVQ, instead of scalar quantizing the scalar difference values (as in DPCM), vectors of difference values (difference tiles) are vector quantized and their codeword indices transmitted. At the receiver, the indices are used to look up the reconstruction difference vector which is then added to the predicted difference vector.

Because the VQ coder uses the FSCL derived codebooks, over a large sample of images the codewords are utilized relatively equally. Consequently, fixed length codes are used and the codec does not need to use synchronizing codes. Furthermore, as discussed earlier, by arranging the codewords so that codewords which are Hamming-close are also distortion-close we can achieve a significant level of error insensitivity, as the results in Section 5 clearly show.

5 Results

Table 1 shows the MSE obtained when DVQ codec using a FSCL codebook was used to compress eight images. For comparison, the MSE for a version of a DPCM codec, as described in [10], is also given. As can be seen, the DPCM codec yields MSEs which are lower than those of the DVQ codec. However, as can be seen in Figures 4 and 5 the images are virtually indistiguishable (these images are indistinguishable at full resolution, as well). Furthermore, as shown in Figure 6, the DVQ codec is significantly more robust to channel errors. In the left picture of Figure 6 line resynchronization was used in an attempt to minimize errors to one line of the image. Note that, even with resynchronization, the method results in truncated lines. Furthermore, when errors occur in the synchronizing codeword, lines are missed entirely.

6 Conclusions

We have presented an ANN based DVQ codec which is well suited for use in data compression applications such as Digital Video. There are three novel aspects of this work. First, Differential Vector Quantization is used to preserve edge features. Second, a new adaptive algorithm, known as Frequency-Sensitive Competitive Learning, is used to develop a vector quantizer codebook that eliminates the need for Huffman coding. Finally, in order to realize real time performance, a custom VLSI ASIC is being constructed to perform the associative memory functions needed in the vector quantization algorithm. The resulting codec exhibits greater compression and superior performance against channel bit errors than methods that use variable length codes.

Acknowledgments

Support for this research was provided by grants from the NASA-Lewis Research Center. We would also like to express our gratitude to Ken Adkins, Matt Carbonara, Surajit Chakravarti, Metin Demirci, Jim Fowler, and Rich Kaul.

7 References

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

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

[3] 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.

[4] 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, no. 8, 1990.

[5] 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.

[6] A. Buzo, A. H. Gray, Jr., R. M. Gray, and J. D. Markel, "Speech Coding based upon Vector Quantization," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-28, pp. 562–574, October 1980.

[7] S. P. Luttrell, "Derivation of a Class of Training Algorithms," *IEEE Transactions on Neural Networks*, vol. 1, pp. 229–232, June 1990.

[8] T.-C. Lee and A. M. Peterson, "Adaptive Vector Quantization Using A Self-Development Neural Network," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 8, 1990.

[9] P. Chen, "The Neural Shell: A Neural Networks Simulator," Master's thesis, The Ohio State University, March 1989.

[10] M. J. Shalkhauser and W. A. Whyte, Jr., "Digital CODEC for Real-Time Processing of Broadcast Quality Video Signals at 1.8 Bits/Pixel," Technical Report NASA TM-102325, National Aeronautics and Space Administration, Lewis Research Center, Cleveland, 1989. Prepared for Global Telecommunications Conference sponsored by IEEE, Dallas, Texas, Nov 27-30, 1989.

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

Table 1: MSE of NASA Codec and FSCL Differential Vector Quantization Algorithms

Picture	*Bits Per Pixel (BPP)*			*MSE: No Errors*			*MSE: BER = 1/1000*		
	codec	dvq 128	dvq 256	codec	dvq 128	dvq 256	codec	dvq 128	dvq 256
bird	2.56	1.75	2.00	6.5	20.2	14.5	2152	70.2	71.6
everest	2.17	1.75	2.00	5.0	16.8	12.6	981	44.1	34.3
fruity†	2.30	1.75	2.00	5.9	16.5	11.9	604	67.1	71.0
hall	2.97	1.75	2.00	7.7	17.8	12.7	1010	71.1	63.0
kitty†	2.75	1.75	2.00	5.9	14.0	10.1	964	65.7	47.9
lenna	2.37	1.75	2.00	5.3	11.4	8.3	1213	47.7	34.6
mandril	4.05	1.75	2.00	13.1	89.0	66.4	3107	305.7	201.1
plane†	2.09	1.75	2.00	4.6	13.9	10.0	1113	56.6	61.6
scene†	2.87	1.75	2.00	7.9	22.6	16.3	1993	75.7	75.7
sf†	2.89	1.75	2.00	9.5	57.1	41.9	3008	159.9	139.7

†These images were used in training the vector quantizer.
Error MSE's averaged over 3 trials.

Figure 4: Original image (left) and reconstructed image using the NASA codec algorithm (right)

Figure 5: Reconstructed image using differential vector quantization, FSCL with 128 codewords (left) and 256 codewords (right)

Figure 6: Reconstructed image with a noisy transmission channel (1 error per 1000 bits), NASA codec algorithm (left) and differential vector quantization using FSCL with 256 codewords (right)