Multiple Stage Vector Quantization Using Competitive Learning

Bruce E. Watkins and Murali Tummala
Department of Electrical & Computer Engineering
Naval Postgraduate School, Monterey, CA 93943

Abstract

Neural networks have recently been applied to the task of vector quantization (VQ) for the coding of speech and image data. The frequency sensitive competitive learning (FSCL) algorithm is one of the most successful techniques applied to vector quantization. The FSCL algorithm displays a substantial computational advantage over the existing techniques such as the Linde, Buzo and Gray (LBG) algorithm for vector quantizers with small codebooks, but unfortunately FSCL requires an excessive amount of training for vector quantizers with large codebooks. We present a possible solution to this problem through the application of the multiple stage vector quantization (MSVQ) technique to the FSCL vector quantizer for use in the coding of image data. By combining a number of smaller code books which can be trained quickly using the FSCL VQ, the MSVQ allows us to form a large effective codebook size which enables us to substantially reduce the computational cost and improve performance. The MSVQ technique also allows us to implement a large codebook with a much smaller storage requirement than if the same size codebook were implemented using the basic FSCL algorithm.

1 Introduction

The ability of competitive learning neural network algorithms to self organize a large data set into categories makes vector quantization a natural application for these algorithms. A competitive learning algorithm called self organizing map (SOM) was first introduced by Kohonen [1]; SOM assumes that the input data vectors are uniformly distributed and thus results in under utilization of output nodes when the input statistical distribution is of a different kind. Another technique termed *adding a conscience to competitive learning* is presented in [2], which introduces a bias term into the network weight update equation. The bias term effectively modifies the weight update equations to penalize the output nodes that have won the competition frequently. This produces a very uniform output node utilization while converging quickly. A variation of the *conscience* technique is frequency sensitive competitive learning (FSCL) algorithm [3]. In FSCL, the distance between, d_i, between the input vector and the output node weight vector is modified by: $d_i^* = d_i g(u)$, where u_i is the number of times the output node i has won the competition and g is termed the fairness function with $g(u) = u$ in most cases. Over many training iterations, the effect of this modification is a remarkably even node utilization. Compared to the *conscience* method, FSCL updates only one set of weights for each input vector, requires only one set of distance calculations, and thus converges faster than the *conscience* method.

The FSCL has proven to be a particularly successful algorithm for vector quantization [3]. This algorithm uses a neural network to produce results almost identical to that of LBG algorithm [4] which has been shown to be optimal. Figure 1 shows a comparison of the number of training iterations required by the FSCL and LBG algorithms. For a small codebook size, the FSCL has a sizable computational advantage whereas for larger codebook sizes, the LBZ is more efficient than FSCL. The major shortcomings of FSCL are twofold: for a simulation implementation, the neural network requires an excessive amount of training to meet convergence for a codebook of practical size (see figure 1); for a hardware implementation, a network of sufficient size for good performance would require a prohibitively large number of processing elements. In this work, we have attempted to address both these problems by proposing an algorithm which can reduce the computational cost of training the neural network and also reduce the number of processing elements required for implementation. Results of performance comparison between the FSCL and the proposed algorithm are also included.

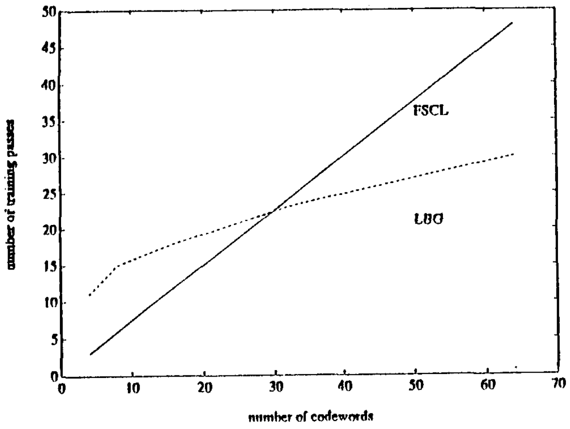

Figure 1: Comparison of the number of training passes required for LBG and FSCL algorithms.

2 Multiple Stage Vector Quantization

The FSCL based vector quantizers are considered optimal here in two senses: Firstly the codebook formed produces the minimum mean squared error (MSE) possible for the training data utilized, and secondly the encoder always selects the codeword corresponding to the vector which produces the least distortion for any given input vector. An algorithm of this type is called full search vector quantization (FSVQ), and it must calculate a number of distortions equal to the size of the codebook for each vector processed. Consequently, this property makes full search codes impractical except for the case of small codebooks.

We now consider an algorithm that produces codes which are suboptimal in both senses mentioned above; however, these algorithms produce codebooks which have a structure that dramatically reduces the computational effort required for a given codebook size. Here, we examine the application of a technique termed multiple stage vector quantization (MSVQ) to the basic FSCL vector quantizer [5]. This technique forms a large effective codebook by combining several smaller codebooks in a linear structure. Figure 2 shows a schematic diagram of the MSVQ algorithm. In this case, each vector quantizer of the structure is a FSCL vector quantizer with a codebook of size n, and the structure consists of m levels. An m level MSVQ can be described by the m-tuple $\mathbf{R} = (R_1, R_2, \ldots, R_m)$, where R_i is the number of bits used to encode the error at level i of the MSVQ, and the effective codebook size is $\Pi_{i=1}^{m} R_i$. The first level of the MSVQ is just a small FSCL vector quantizer. The input vector, $\mathbf{x}$, is applied to the vector quantizer at level one, and the first estimate, $\mathbf{y}_1$, is produced along with the first R_1 bits of the channel codeword, $\mathbf{u}_1$. Next, the first error vector is formed by taking the vector difference, $\mathbf{e}_1 = \mathbf{x} - \mathbf{y}_1$. This error vector is then applied to the size 2^{R_2} vector quantizer at level two, which produces an estimate of the error vector, $\hat{\mathbf{e}}_1$, and the next R_2 bits of the channel codeword. So, at the second level, our estimate of the input vector is the vector sum $\mathbf{y}_2 = \mathbf{y}_1 + \hat{\mathbf{e}}_1$. In the following stages, we continue to form an error vector from the previous stage and use a FSCL vector quantizer to encode this error. Each stage produces an estimate for the error and a portion of the channel codeword. At the last stage, the error vector $\hat{\mathbf{e}}_{m-1}$ is encoded, and the final estimate of the input vector is available by performing $\mathbf{y}_m = \mathbf{y}_1 + \hat{\mathbf{e}}_1 + \hat{\mathbf{e}}_2 + \ldots + \hat{\mathbf{e}}_{m-1}$, and the full channel codeword obtained by catenating $\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m)$.

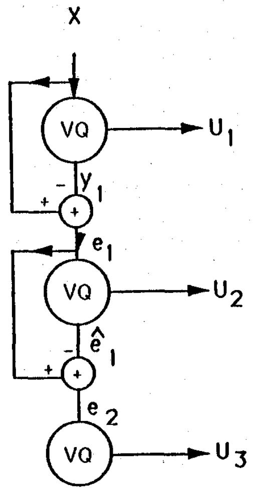

Figure 2: Structure of the multiple stage vector quantization scheme.

The training of the MSVQ proceeds one level at a time. We apply the original data set to FSCL vector quantizer at the first level until convergence is obtained. Then, we pass the data through the trained vector quantizer and compute the error vector between each input vector and the closest code vector. This forms a new data set, which is a collection of the first stage errors. This first stage error data set is then applied to the vector quantizer on the second level until convergence is obtained; then, it is applied a final time to compute the second stage error vectors. This continues until the last stage has been trained. MSVQ provides an initial estimate at the first level and produces a better estimate at each level by continuing to add smaller and smaller correction terms in a way similar to the method of successive approximations. Each of these corrections is a result of performing vector quantization on the error subspace of the preceding level.

For encoding, MSVQ requires $\sum_{i=1}^{m} 2^{R_i}$ distance calculations which is far fewer than the 2^R required for FSVQ. Table 1 shows the total number of distance calculations which must be calculated for several examples. This dramatically reduces both storage requirements and the load on the channel from transmitting codebook updates. Table 2 shows the extra load on the channel for each algorithm assuming that the codebook is updated with each frame. The advantage of MSVQ in this regard for large codebooks is apparent.

Table 1: Distance Calculations Required

	Distance Calculations	
Codebook Size	FSVQ	MSVQ
16	16	8
64	64	16
512	512	24

Table 2: Channel Load of Codebook Transmission (bits/pixel)

	Channel Load (bits/pixel)	
Codebook Size	FSVQ	MSVQ
16	0.008	0.004
64	0.047	0.012
512	0.563	0.026

Figure 3 shows that MSVQ provides a huge computational advantage, especially for large codebooks. The MSE performance for each algorithm is compared in figure 4. While it is clear that MSVQ produces a suboptimal codebook, the large gain in computational costs allows us to use larger effective codebooks and actually to obtain an improvement in performance as will be shown in an example later. MSVQ provides an extremely simple structure which would require only a small number of processing elements and would make hardware implementation much simpler. Finally, because MSVQ uses only one vector quantizer per level, the algorithm vastly reduces the amount of storage required for simulation and decreases the load on the transmission channel due to codebook transmission.

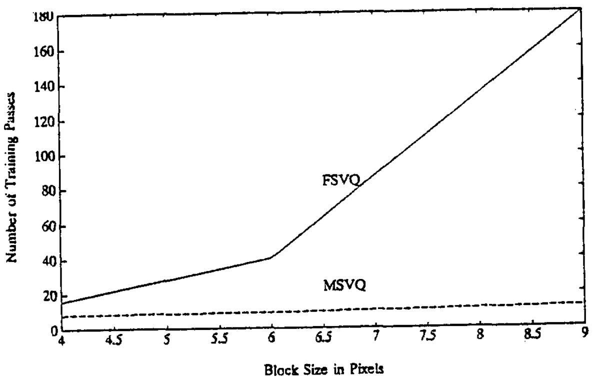

Figure 3: Computational cost.

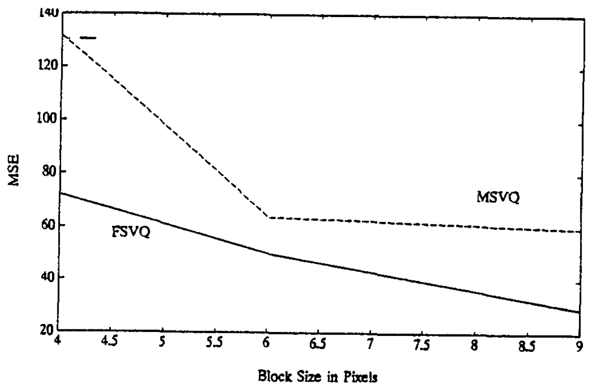

Figure 4: Performance comparison of FSCL and MSVQ algorithms.

3 Simulation Results

The simulations were conducted on a single test image of 256 x 256 pixels. This image was divided into blocks of various sizes chosen to yield a data rate of 1 bit/pixel for each reconstruction using a variety of codebook sizes. Figure 5 shows the results using MSVQ on a 256 x 256 image using a block size of 3 x 2 with $n = 8$ and $m = 2$ for an effective codebook size of 64. This corresponds to a data rate of 1 bit/pixel and the mean square error (MSE) is 63.68. Figure 6 shows the same image coded with a block size of 2 x 2 using the original FSCL algorithm and has a MSE of 71.01. The MSVQ required only 9 passes through the image to form the codebook versus 16 passes for the original FSCL which shows a substantial computational advantage. Also, the MSVQ technique allowed us to implement a codebook of size 64 with a storage requirement of only 16 code vectors. This increased codebook size provides the increase in performance that we see in the example presented (in figures 5 and 6).

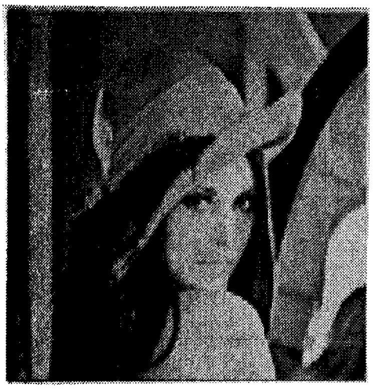

Figure 5: Reconstructed image using MSVQ algorithm (1 bit/pixel).

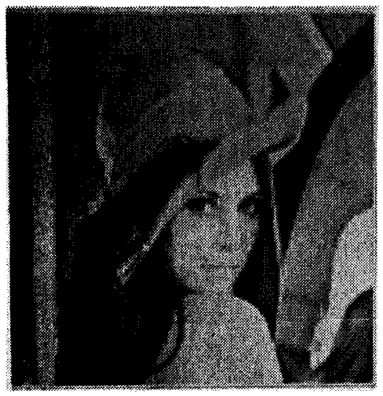

Figure 6: Reconstructed image using FSCL algorithm (1 bit/pixel).

For large codebooks, the MSVQ algorithm provides a huge computational advantage over the FSVQ method. It also provides an extremely simple structure which would require only a small number of processing elements and would make hardware implementation much simpler. Finally, because MSVQ uses only one vector quantizer per level, the algorithm vastly reduces the amount of storage required for simulation and decreases the load on the transmission channel due to codebook transmission. The performance of the MSVQ falls short of the standard FSVQ algorithm. The reason for this suboptimality can be seen in the structure of the MSVQ in which we form the codebook for the next level using the error vector instead of the original input vector. The relatively poor performance of the MSVQ algorithm can thus be attributed to using the probability density function of the error vectors in the subsequent stages which is different from that of the original input vectors.

References

[1] T. Kohonen, "Self Organization and Associative Memory," *Springer Verlag*, 1984.

[2] D. DeSieno, "Adding a Conscience to Competitive Learning," Proc. *IEEE Int. Conference on Neural Networks*, pp. 1117–1124, July 1988.

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

[5] B.-H. Juang, " Multiple Stage Vector Quantization for Speech Coding," In *Proc. 1982 Int. Conf. Acoustics, Speech, Signal Processing*, pp. 597–600, Paris, April 1982.