Neural Tree Structured Vector Quantization

Eric Wan Paul Ning Bernard Widrow
Stanford University Department of Electrical Engineering, Stanford, CA 94305-4055

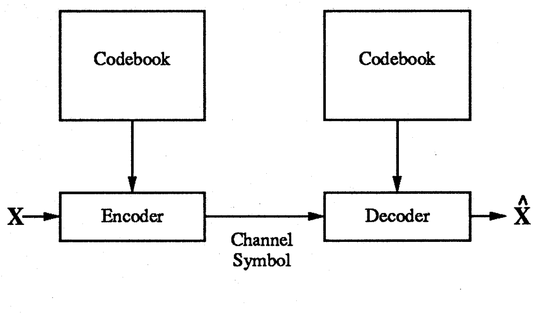
Abstract

In this paper we present a new method for vector quantization design and implementation using a tree structured artificial neural network. Each node in the tree consists of a neural network which successively learns to partition the input space. While the structure is evaluated for use in image compression, the NTSVQ can be generalized as a means of data compression wherever vector quantization is appropriate. In addition, the adaptive tree structure should have many applications in related pattern recognition problems.

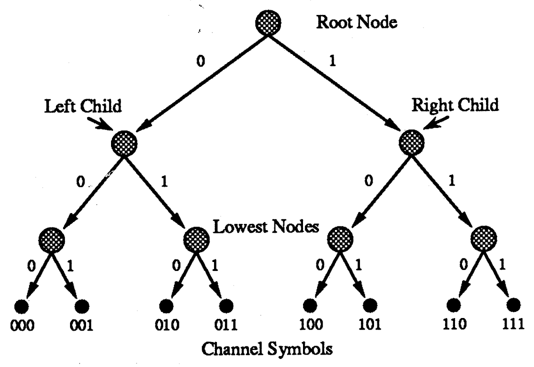
1 Introduction

In recent years, vector quantization (VQ) has become an increasingly popular form of data compression (see [1] for a review). A traditional vector quantizer consists of an encoder, decoder, and codebook as shown in Figure 1a. Given a vector to be quantized, the encoder finds in a codebook the codeword which minimizes the distortion between it and the vector to be quantized. The binary index of the codeword, called the channel symbol, represents this vector and can be processed, transmitted, or stored. A decoder having an identical codebook uses the channel symbol to locate the corresponding codeword for the reproduction vector. For a codebook of k codewords and a vector dimension of N, the bit rate is defined as $log_2(\text{k})/N$ bits per vector component. The codewords themselves can be stored to an arbitrary accuracy and do not figure into the bit rate. The process of encoding and decoding is straightforward. The more difficult task, however, is the design of the codebook which minimizes the overall distortion of the data to be quantized. The standard VQ design technique, the LBG algorithm [2], utilizes training vectors to establish the codebook.

In a tree structured VQ, encoding is achieved through a series of binary decisions in which the input space is successively partitioned. Referring to Figure 1b, the input is first evaluated by the root node, which is at the top. Depending on the result of this initial test, the input vector is then passed to either the left or right child of the root. Each child, in turn, is another node which makes binary decisions to continue the classification process. Each node in a VQ tree actually consists of a tiny VQ with a codebook of size two. The binary decision at the node consists of determining which of the two reconstruction vectors incurs the smaller error. The effective codebook of the tree VQ is the union of the codebooks of the lowest level nodes. For a binary tree with L levels, the total number of classes or codewords is 2^L. Reconstruction vectors used by any node above the bottom do not appear in the final codebook but only determine the search path down the tree. This binary path through the tree corresponds directly to the actual channel symbol. Since only one comparison is needed at each level, the computations required to search a tree grow only with the logarithm of the codebook size (a great savings over the fullsearch VQ which grows linearly with codebook size).



(a)



(b)

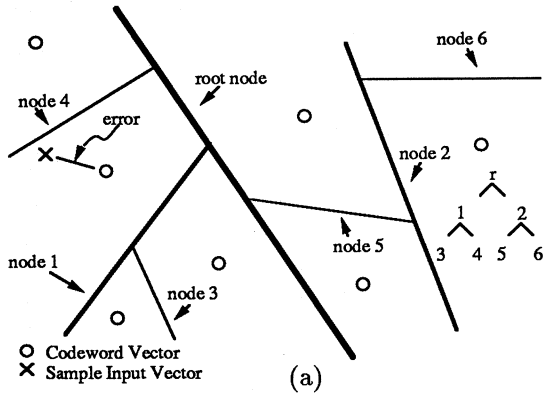
Figure 1: (a) Traditional VQ System, (b) Binary Decision Tree Encoder

The process of quantizing an N-dimensional vector amounts to partitioning N-dimensional Euclidean space into k partitions. The centroid of each partition represents each codeword. A vector falling into one of these partitions is represented by the centroid of the partition. The error as a result of quantization can be viewed as the distance between the actual vector and the codeword. An example of a 2-dimensional vector space partitioned by a binary decision tree is shown in Figure 2a.

2 Learning in the Decision Tree

We now present a method of implementing a tree structured VQ based on neural network techniques. Each node in the decision tree of Figure 1b contains a small adaptive neural network capable of binary decision making. The architectures of the nets are the same for all the nodes. The weights are different from node to node, depending on learning experience.

The typical neural network for each node is shown in Figure 2b, The simple 2-layer neural net can be considered as a two-codeword vector quantizer. The first layer is an encoder and the second layer is a decoder. The values of the encoder weights determine the partitioning of the input space. For a single layer of encoder weights, this partition is a simple hyperplane at each node. While additional encoder layers would allow for non-linear partitions, the optimal partition for a two codeword codebook is always a hyperplane. The sign of the sigmoid output at the hidden channel layer determines the binary decision for the node. The collective binary decisions down the tree formulate the full channel symbol.



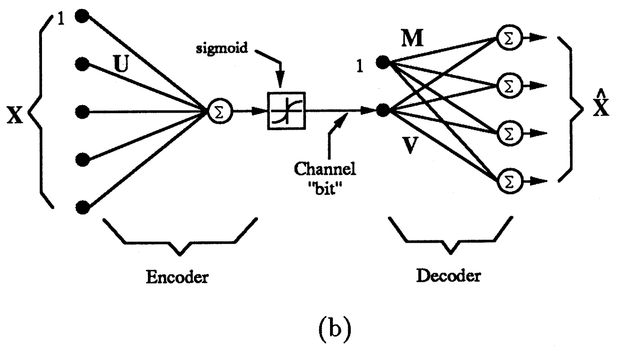


Figure 2: (a) Tree Partitions, (b) A Node's Internal Neural Network

Node Adaptation Referring to Figure 2b, training is effected using the backpropagation algorithm [3] to cause the decoder output $\hat{\mathbf{X}}$ to be a least squares reconstruction of the input $\mathbf{X}$. The sigmoid is necessary for backpropagation. However, after convergence of the two layers, the sigmoid must be replaced by a hard binary quantizer. This produces an actual bit at the encoder output and drives the second layer with a binary signal. The decoder weights are then readapted using simple LMS [4] to once again minimize the mean square error. With the encoder weights fixed, and the partitions set, the decoder outputs will converge to two possible reconstruction vectors $\hat{\mathbf{X}}$ which will become the centroids of each partition. Thus, the system of Figure 2b is a self-contained self-organizing classifier which is used here as a binary vector quantizer. The tree itself forms a more sophisticated vector quantizer that also learns without supervision.

Tree Adaptation Since nodes will only need to make decisions on inputs which are passed to them from higher nodes, it makes sense to adapt their weights on only those vectors. Thus to adapt the entire tree we use the following training cycle :

1. Adapt the root node to the current vector in the training set.
2. Depending on the side to which the root node classifies the input vector (based on the sign of the channel bit), select either the left or right child to receive the input.
3. Perform steps 1 and 2 for the selected node and continue until the bottom of the tree is reached.
4. Repeat the cycle for a new input chosen randomly from the training set.

In this way, for any given input training vector, only L nodes in the tree are adapted corresponding to that input's classification path. Multiple passes through the training set are performed until the weights converge or a suitable VQ distortion level is achieved.

3 Analysis

A study of an individual node reveals how partitions are formed and provides insight into the nature of the reconstruction vectors. Referring to Figure 2b, let the input $\mathbf{X} = [1, x_1, ..., x_N]^T$ be a sample vector plus bias. $\mathbf{U} = [u_0, u_1, ..., u_N]^T$ is the encoder weight vector, $\mathbf{V} = [v_0, v_1, ..., v_N]^T$ is the decoder weight vector, and $\mathbf{M} = [m_0, m_1, ..., m_N]^T$ is the decoder bias weight vector. The output reconstruction vector is then

$$\hat{\mathbf{X}} = \mathbf{V}f(\mathbf{X^T U}) + \mathbf{M} \tag{1}$$

and the error due to reconstruction is

$$\mathbf{E} = \mathbf{X} - \hat{\mathbf{X}} = \mathbf{X} - \mathbf{V}f(\mathbf{X^T U}) - \mathbf{M} \tag{2}$$

For a performance measure we consider the total mean squared error $\xi = E[\mathbf{E^T E}]$. The optimal solution for the weight vectors which minimizes the overall mean squared error can be found by setting the gradients of ξ with respect to the various weight parameters to zero and solving a set of nonlinear matrix equations. Unfortunately, no closed form solution exists. However, by assuming operation in the linear region of the sigmoid (i.e. $f(x) \approx x$), several interesting results may be obtained. With this assumption

$$\xi = TR[\mathbf{R}] + 2\mathbf{U^T \bar{X} V^T M} + \mathbf{U^T R U V^T V} - 2\mathbf{M^T \bar{X}} + \mathbf{M^T M} - 2\mathbf{U^T R V} \tag{3}$$

where $\mathbf{R} = E[\mathbf{XX^T}]$. By taking the gradient with respect to each of the weight vector, $\mathbf{U}, \mathbf{V}$, and $\mathbf{M}$, it can be shown that the following conditions must hold at a minimum

$$\mathbf{M} = \bar{\mathbf{X}} \qquad \mathbf{U} = \frac{\mathbf{V}}{\mathbf{V^T V}} \tag{4}$$

$$\mathbf{RV} = \lambda \mathbf{V} \qquad \xi_{min} = TR[\mathbf{R}] - \mathbf{U^T R V} \tag{5}$$

This implies that whenever $\mathbf{V}$ is an eigenvector of $\mathbf{R}$, and the conditions in (5) are met, we are at a local minimum. Thus there are up to N local minima, where N is the dimension of $\mathbf{X}$. The optimal global solution is achieved when $\mathbf{V}$ is the maximal eigenvector of $\mathbf{R}$. The globally minimum expected mean squared error is then

$$\xi_{min} = TR[\mathbf{R}] - \lambda_{max} \tag{6}$$

and the optimal partition defined by $\mathbf{X^T U} = 0$ is the hyper-plane normal to the maximal eigenvector of $\mathbf{R}$. While this result is only valid for the linear case, it should reflect how partitions are initially formed during the early stages of adaptation with the sigmoid. The nature of the final partitions with the full non-linearities involved is still under investigation.

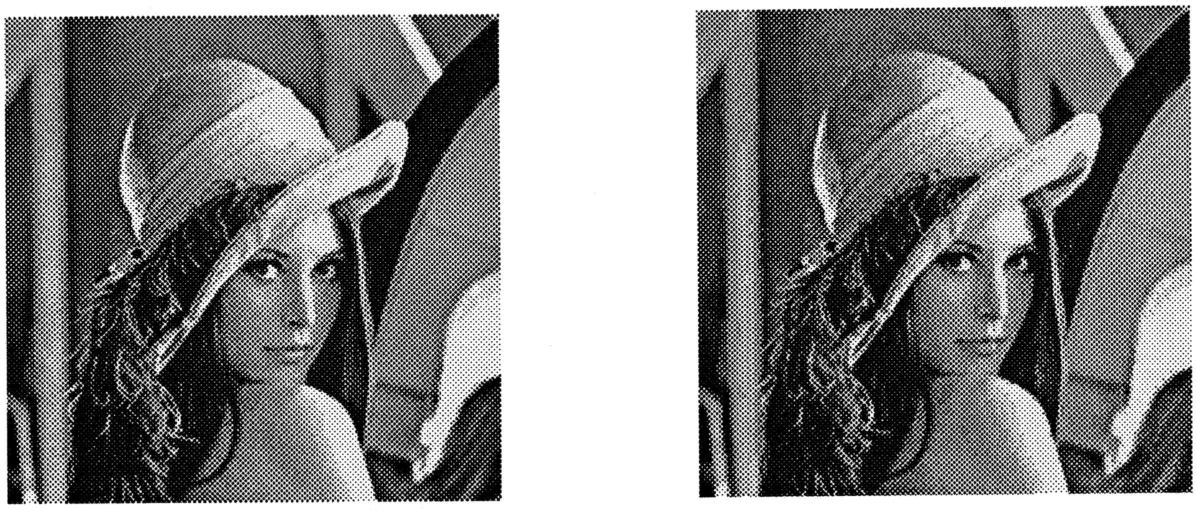
4 Applications To Image Compression

For image compression, the input vectors to the VQ system are obtained by dividing up images into sets of non-overlapping square blocks of pixels. Thus, each image provides a large number of input vectors. Training vectors can be obtained from several representative images.

A sample image compressed to 2 bits per pixel with a 2x2 block size is shown in Figure 3. The original 8 bit per pixel image, with pixel intensities ranging from 0 to 255, is reconstructed with a root mean square error (per pixel) of 6.17. The compression ratio is 4:1. The reconstructed image is almost as good as the original.

5 Variations

Philanthropic Trees Traditional decision trees are often described as *greedy* . A parent is designed to do the best it can without any consideration of how its child is doing. The overall performance of a tree may be improved if at any given intermediate node a sacrifice in performance is allowed based on knowledge of how its decision affects its children's performance. Neural TSVQ's offer the unique potential of being able to adapt all nodes in a fully interdependent manner. The error of a child can be weighted into the error of its parent. The parent which attempts to minimize its own mean squared error is now directly influenced by how well its children are doing. The parent, which determines the subset of the training set its children see, in turn affects the expected mean squared error of the children. The details of philanthropic trees are left to [6]. Experiments with error passing have shown improvements in MSE on the order of 10 percent. It is our view that various error passing schemes which allow one to adapt the tree as a fully interdependent unit will ultimately provide many advantages over traditional decision trees.



(a) (b)

Figure 3: **Lena (a) Original (b) Compressed**

Pruned Trees Additional improvements to Tree VQs can be obtained by using variable length trees. One can prune selected nodes to cut the tree short along paths which include inefficient nodes. This lowers the average bit rate of a coder without undue loss in mean square error. Equivalently, this procedure can yield lower distortion for any given average bit rate by first growing a uniform tree at a higher bit rate and then pruning back to the desired rate. Pruning results in trees with classification paths similar in nature to entropy based codes and can outperform even full search quantizers. An "optimal" pruning technique can be found in [5].

6 Conclusion

In this paper we have presented a new method for vector quantization which combines decision tree techniques with neural network techniques. While we have concentrated on data compression, the neural tree structure can be applied to a variety of decision problems. Inputs to the network may, for example, correspond to segments of speech. The tree would develop in such a way that decision paths identify phoneme classes. A study of this structure for use in character recognition is already under investigation and will be reported in a forthcoming paper. Finally, other possible variations include alternate error passing algorithms, the inclusion of additional layers in the encoder and decoder, and the use of teacher directed learning. A detailed discussion of this structure can be found in an expanded version of this paper [6].

References

[1] R.M. Gray, "Vector quantization," *IEEE ASSP Magazine*, vol. 1, no. 2, pp. 4-29, April 1984.

[2] Y. Linde, A. Buzo, and R.M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Comm.*, vol. COM-28, pp. 84-95, Jan. 1980.

[3] D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, The MIT Press, Cambridge, MA, 1986.

[4] B. Widrow and M.E. Hoff, "Adaptive switching circuits," *1960 IRE WESCON Convention Record*, Part 4, pp. 96-104, August 1960.

[5] P.A. Chou, T. Lookabaugh, and R.M. Gray, "Optimal pruning with applications to tree-structured source coding and modeling," 1987. *IEEE Trans. Information Theory*, to appear.

[6] E.A. Wan, P. Ning, "Neural Tree Structured Vector Quantization" 1988, *Stanford report.*