EXHIBIT BB

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

Adaptive Vector Quantization Using a Self-Development Neural Network

TSU-CHANG LEE AND ALLEN M. PETERSON, LIFE FELLOW, IEEE

Abstract—A novel neural network model, called SPAN (space partition network), is presented. This model differs from most of the currently seen neural networks in that it allows a network to adapt its structure by adding neurons, killing neurons, and modifying the structural relationships between neurons in the network. An adaptive vector quantization source coding system based on SPAN is proposed. The basic idea is to use SPAN as *an active codebook* that can adapt its structure to follow the source signal statistics. The major advantage of using SPAN as the codebook of a vector quantizer is that SPAN can capture the local context of the source signal space and map onto a lattice structure. A fast codebook searching method utilizing the local context of the lattice is proposed and a novel coding scheme, called the *path coding method*, to eliminate the correlation buried in the source sequence is introduced. The performance of our proposed coder is compared to an LBG coder on synthesized Gauss–Markov sources. Simulation results show that, without using the path coding method, SPAN yields a similar performance to an LBG coder; however, if the path coding method is used, SPAN displays a much better performance than the LBG for highly correlated signal sources. Because the training method is smooth and incremental, SPAN is suitable as the basis for an adaptive vector quantization system.

I. Introduction

In recent years, there has been much interest focused on vector quantization for different physical signal sources, especially speech and image signals [15], [27], [11]. The driving force behind this trend is listed in the following.

- According to Shannon's rate-distortion theory, a better performance is always achievable in theory by coding a block of signals (i.e., vectors) instead of coding each signal individually [31], [32], [10], [2], [35], [16], [14].
- Technological improvements, especially progress in VLSI computation ability and memory capacity, make more sophisticated coding/decoding systems possible.
- As technological enhancement advances, the requirements placed on communication subsystems become more and more demanding. Several key driving technologies strongly require data compression techniques; among these are high-definition TV and integrated service data networks.
- The nature of the signals to be represented in computers moves from artificial signals (like symbols, texts, etc.) to signals closer to those in the physical world (e.g., sound, images, etc.), which tend to be more unpredictable and hard to characterize analytically. Because of this, flexible and adaptable signal representation schemes will become more and more important in the near future.

A. Vector Quantization Problem

Vector quantization can be viewed as a mapping $\mathcal{Q}$ from a k-dimensional vector space R^k into a finite subset W of R^k

$$\mathcal{Q}: R^k \to W \tag{1}$$

where $W = \{w_i \mid i = 1, 2, \cdots, M\}$ is the set of reproduction vectors and M is the number of vectors in W. Each w_i in W is called a *codeword* and W is called the *codebook* for the vector quantizer. For every source vector x, a codeword w_j in W is selected as the representation for x. This process is called the *quantization phase* (or the *codebook search phase*) of the vector quantizer, denoted by $\mathcal{Q}(x) = w_j$. Then, the codeword w_j is represented by some symbols (normally the address of the codeword in the codebook) and transmitted through the channel. This process is called the *encoding phase* of the vector quantizer. On the other side of the channel, the received symbols are used to select the codewords from the codebook to reproduce the source signals. This process is called the *decoding phase* of the vector quantizer. The average number of bits required to represent the symbols in the encoding phase is the *rate* of the quantizer, and the average quantization error between input source signals and their reproduction codewords is the *distortion* of the vector quantizer. Increasing the number of codewords in the codebook can decrease the distortion of a vector quantizer and, normally, will increase the rate also. One major concern for vector quantizer design is the trade-off between distortion and rate.

Linde, Buzo, and Gray [25] pointed out that the *necessary condition* for an optimum codebook is that each codeword is the centroid of the set of source signals it is representing. This suggests a *fixed point method* to generate the codebook. The idea is shown in the following.

- The codewords partition the source signal space into regions according to the closest neighbor relationship.
- The centroids of the partitions generate another set of codewords.

Manuscript received September 5, 1989; revised February 15, 1990. This paper was supported in part by the NASA CASIS program under Contract NAGW 419 and by a gift fund from the Ford Aerospace and Communication Company. This paper was presented in part at the 6th IEEE International Workshop on Microelectronics and Photonics in Communications, New Seabury, Cape Cod, MA, June 7-9, 1989.
The authors are with STAR Laboratory, Department of Electrical Engineering, Stanford University, Stanford, CA 94305-4055.
IEEE Log Number 9036210.

Exhibit BB Page 515

• The above process continues until the codebook converges to a fixed point in the solution space of codebooks.

In most cases, it seems that the codebooks generated by this algorithm display at least local op··· ·· m distortion rate performance if the algorithm converges during the codebook generation process [25], [14]. This algorithm is commonly referred as the *LBG algorithm* or *generalized Lloyd algorithm* because it is a generalized version of the scalar quantization algorithm proposed by Lloyd [26] in 1957.

A second important issue in vector quantizer design is codebook search efficiency. The major concern in this aspect is how to develop efficient ways of searching through the codebook to find the optimum reproduction codewords for the source signals. This is normally done by incorporating some structure irto the codebook to help the search process. For examp!~, if we incorporate a tree structure into the codebook, the search time is a logarithm function of the size of the codebook [5], [14].

B. Vector Quantization Using Neural Network Paradigm

There are several important problems not addressed by the LBG algorithm.

• Some codewords might be underused. In the LBG algorithm, all the codewords share the same rate (i.e., all of them are represented by the same number of bits), but not each of them contributes equally to the average system distortion. That means some codewords are not sharing the same *representation load* as others. In the extreme case, some codewords might never be accessed.

• If the statistics of the source signal change, how does one modify the codebooks on both sides of the channel to reflect the changes? The desirable adaptation should be in a smooth and incremental manner.

Several researchers tried to tackle the problems above using neural network paradigms. For example, Kohonen [19] proposed a self-organization neural network vector quantizer that incorporates local interactions between codewords to form a topological feature map that *topologically sorts* the codewords on a lattice. Kohonen's vector quantizer has also been used to code images and shows a similar performance to the LBG algorithm [28], [29], [1]. Krishnamurthy [1] proposed using an access frequency-sensitive distortion measure to select the codewords in order to avoid the codebook underuse problem.

In this paper, we describe a self-development neural network codebook that can grow from scratch to follow the statistics of source signals, and to capture the local context of the source signal space to map onto the structure of the network. When the statistics of the source signals change, the network can dynamically modify its structure to follow the change. We then introduce a coding scheme that utilizes the context of the network to guide the codebook search process (hence, to enhance the searching efficiency) during the quantization phase and to eliminate the correlation between adjacent signal sources (hence, to reduce the rate) during the encoding phase. The last section describes the system structure of a fully adaptive source coding/decoding system.

II. Neural Network Codebook

A. Basic Framework

The codebook of our proposed neural network coder/decoder is a lattice-structured neural network called SPAN (space partition network). Each codeword in the codebook is represented by a neuron. The neurons in the network are arranged in a lattice structure with each neuron assigned a position in the lattice. In general, a k-dimensional lattice $\mathcal{L}$ is defined as $\mathcal{L} = \{\sum_{i=1}^{k} b_i v_i \mid (b_1, b_2, \cdots, b_k) \in Z^n, Z \textit{ is the set of integers}\}$, where $v_i, v_2, \cdots, v_k$ are a set of linear independent vectors called the *primitive translation vectors* (PTV's) of the lattice [18]. With the PTV's defined, any point in the lattice can be represented by an index vector $(b_1, b_2, \cdots, b_k)$. In this paper, we are interested only in rectangular lattices;[1] however, the ideas can be extended to more general lattices.

Each neuron i has a neighborhood $\mathfrak{N}_i$, which is a set containing a group of neurons around i in the lattice. If $\mathcal{S}$ is the set of neurons and $\mathfrak{N} = \{\mathfrak{N}_i \mid i \in \mathcal{S}\}$ is the set of neuron neighborhoods, then the pair $\{\mathcal{S}, \mathfrak{N}\}$ forms a graph in the usual sense.

The input signals to the network are the source signals to be represented by the network. Each neuron in the network contains a group of variables, constituting the state of the neuron. Every neuron updates its state according to the input signal and the context of neuron states within its neighborhood. One important state variable of a neuron, called the *input weight vector*, is the reproduction codeword for this neuron.

The network dynamics are characterized by the state transition behavior of all the neurons in the network. There are two different levels of adaptation in SPAN.

• *Parameter Level Adaptation:* Take the structure of the network as fixed and adapt the state variables of the neurons in the network. This corresponds to the weight adjustment process in most of the commonly seen neural network models.

• *Structure Level Adaptation:* Adapt the structure of the network by changing the number of neurons and the structural relationship between neurons in the network.

Notice that the general conceptual architecture of SPAN is similar to that of a *cellular automata* [37], [30], [36]. However, the difference between SPAN and traditional cellular automata is that the structure of SPAN is adaptable.

B. Weight Adjustment Process of SPAN

We follow Kohonen's learning algorithm [19], [20], [22] to adjust the input weight vectors of neurons in the network:

[1]In a rectangular lattice, the PTV's form an orthonormal set.

Exhibit BB Page 516

```
for (for each index $n$) do
    fetch source $x[n]$
    select $i$: $\|x[n] - w_i[n-1]\| \le \|x[n] - w_k[n-1]\|, \forall k \in \mathcal{S}$
    for ($\forall j \in \mathfrak{N}_i$) do
        $w_j[n] = w_j[n-1] + \alpha\phi[n-1](\|l_j - l_i\|)(x[n] - w_j[n-1])$
    end
end
```

where $x[n]$ is the input source signal of the network at time n; $w_j[n]$ is the input weight vector of neuron j at time index n; $\mathcal{S}$ is the set of neurons; α, called the *learning rate*, is a constant used to control the rate of convergence in the learning process; l_i is the position of neuron i in the lattice, and $\phi[n](d)$, called the *spatial mask*, is a time-varying function used to control the range of influence of a neuron to other neurons in the lattice. Kohonen [20] showed that, if $\phi[\,](\,)$ is selected in such a way that the influence range is wide in the beginning and gradually decreases to a minimum value close to the end of the training process, then both fast convergence and final representation accuracy of the input weight vectors to source signals can be achieved.

The $\phi[\,](\,)$ used by Kohonen is

$$\phi[n](a) = \begin{cases} 1 & \text{if } a \le R[n] \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $R[\,]$ is a positive definite monotonic decreasing function, the minimum value for $R[\,]$ is 1 for the network to retain organizational capability. In general, $\phi[n](\,)$ can be of the form

$$\phi[n](a) = \begin{cases} g(a/d[n]) & \text{if } a \le R[n] \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where both $g(\,)$ and $d[\,]$ are positive definite monotonic decreasing functions. Notice that $R[n]$ in (3) and (2) defines a time-varying HNS, with the size of neighborhoods decreasing with time. A homogeneous neighborhood system (HNS) is of the form (see Fig. 1)

$$\mathcal{H} = \{\mathcal{H}_i \mid i \in \mathcal{S}\};\ \mathcal{H}_i = \{j \mid \|l_j - l_i\|^2 \le r, j \in \mathcal{S}\} \tag{4}$$

where $\mathcal{S}$ is the set of neurons, l_i and l_j are the lattice positions for neuron i and j, respectively.

Kohonen's learning algorithm creates a topological neighboring relationship-preserving vector quantizer. After enough input training vectors have been presented, the input weight vectors of the neurons will specify clusters or vector centers that sample the input space, such that the point density function of the vector centers tend to approximate the probability density function of the input vectors. Besides, the weights will be organized such that neighboring neurons on the network will tend to have similar input weight vectors, representing the neighboring regions in the input pattern space. In particular, it can be shown that the asymptotic values of the weight vectors will tend to be the weighted centroid of their influence regions. The influence region for neuron i is the region Ω_i in the input pattern space such that, during the training process, whenever an input pattern falls in Ω_i, w_i will be modified. To be more specific, $\Omega_i = \bigcup_{j \in \mathfrak{N}} V_j$, where V_j is the Voronoi region for neuron j.

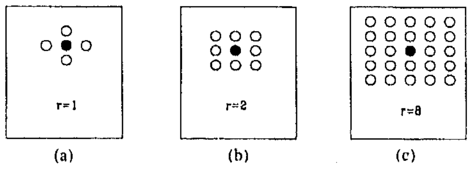


Fig. 1. Homogeneous neighborhood systems (HNS). (a) Minimum HNS, each neighborhood contains only the closest neighbors; (b) $r = 2$ case, each neighborhood includes neighbors no farther than the second closest neighbors; (c) $r = 8$ case, each neighborhood contains neighbors no farther than the fifth closest neighbors.

This idea is formalized in the following theorem (see the Appendix for proof of this theorem).

Theorem 1: Let $y_i(x) = \phi(\|l_i - C(x)\|)$, where l_i is the lattice position for neuron i and $C(x) = l_j$ is the lattice position for neuron j that has the input weight vector closest to the input signal x. For sufficiently small learning rate α, the asymptotical value of the weight vector for neuron i is

$$\hat{w}_i = \lim_{n\to\infty} w_i[n] = \frac{\int_{\Omega_i} p(x)\, y_i(x)\, x\, dx}{\int_{\Omega_i} p(x)\, y_i(x)\, dx} \tag{5}$$

where $p(x)$ is the probability density function for input patterns.

This property makes the network capable of catching the local context of incoming source vectors and thus suggests that the network can be used for the following purposes:

- as an associated memory to restore the original signals from incomplete or degraded input patterns [20];
- as a sequence classier to catch the patterns buried in the context of a sequence of input vectors [21], [34];
- as a context-sensitive VQ to eliminate the correlation between adjacent vectors in the input source signal streams. We will explore this idea further in Section III.

C. Structure Level Adaptation Processes for SPAN

In Kohonen's network model, the structure is usually an $N \times M$ array of neurons. The network tries to catch the statistical distribution of input source signals by adjusting the input weight vectors of neurons in the network. However, the structure of the network is fixed by the network designer in the beginning of the training process. In view of this limitation, one must think of the following questions.

Exhibit BB Page 517

• How do we decide the initial structure of the network, especially when the statistics of the source signals are unknown?

• Even if we can figure out the initial structure of the network, if the statistics of the source signals gradually change with time, how do we modify the structure of the network to reflect the change?

The general version of the above questions poses what we call *the frame problem* of artificial neural networks [24]. To tackle the frame problem of neural networks, we propose adapting the structure of the network by modifying the following:

• the number of neurons in the network;

• the structural relationship between neurons in the network.

Through structure-level adaptation, SPAN can dynamically modify *the frame* that neurons reside in, so that the structure of the network always reflects that of the input pattern space.

1) Neuron Generation Process: As for any vector quantizer, there is a trade-off between quantization error and representation complexity in SPAN. Since increasing the number of neurons in SPAN will decrease the quantization error (or distortion) of the network, we can use quantization error as a measure to determine when to generate new neurons. If a neuron i contributes too much to the average distortion of the network, that means its Voronoi region[2] V_i in the input pattern space is underrepresented by neuron i, hence, a new neuron should be generated to share some of the *representation load* of neuron i, so that the average system distortion can be decreased.

The average system distortion is

$$D = E\left[\|x - \mathcal{Q}(x)\|^2\right] = \int_V \|x - \mathcal{Q}(x)\|^2 p(x)\, dx \tag{6}$$

where $V = \bigcup_{i \in S} V_i$ is the input space.

Whenever the source signal x falls into the Voronoi region V_i, neuron i is selected to represent the input signal, and the vector quantizer replaces x by $\mathcal{Q}(x) = w_i$; hence, we may write

$$D = \sum_{i=1}^{M} \int_{V_i} \|x - w_i\|^2 p(x)\, dx \tag{7}$$

where M is the number of neurons in the network.

The probability that neuron i is selected is

$$P_i = \int_{V_i} p(x)\, dx. \tag{8}$$

[2]As suggested by the name SPAN (space partition network), the neuron input weight vectors partition the input pattern space into regions, called *Voronoi regions*, with each region represented by one neuron. All the points in a region are closer to the input weight vector of the corresponding neuron than any other neurons in the network.

Equation (7) may then be replaced by

$$\begin{aligned} D &= \sum_{i=1}^{M} \left(\int_{V_i} \|x - w_i\|^2 \frac{p(x)}{P_i}\, dx \right) P_i \\ &= \sum_{i=1}^{M} E\left[\|x - w_i\|^2 \mid x \text{ in } V_i\right] P_i \\ &= \sum_{i=1}^{M} d_i P_i \end{aligned} \tag{9}$$

where d_i is the average distortion seen by neuron i.

From (9), we know that the contribution of neuron i to the overall system distortion is $d_i P_i$. This can be used as the measure to guide the neuron generation process. Our proposed criterion for neuron generation is the following.

Suppose the allowable average system distortion is ϵ_d, then neuron i should be split into two neurons if

$$d_i P_i > \frac{\epsilon_d}{M}. \tag{10}$$

Now the question becomes how to measure d_i and P_i dynamically. To handle this, we define an *operational measure* of d_i as

$$\hat{d}_i[n_i^k] = \gamma \hat{d}_i[n_i^{k-1}] + (1 - \gamma) \left\|x[n_i^k] - w_i[n_i^{k-1}]\right\|^2 \tag{11}$$

where n_i^k is the time index when neuron i is the kth time being selected; $\hat{d}_i[m]$ is the distortion measure for neuron i at time index m; and γ, a factor between 0 and 1, is used to control the *effective temporal window size* for the averaging process. Notice that $\hat{d}_i$ is updated only when neuron i is selected, and between n_i^{k-1} and n_i^k, $\hat{d}_i$ stays the same. Notice also that (11) defines a moving average filter with an infinite window size [17]. The input signal to the filter is the process $D_i[k] = \hat{d}_i[n_i^k]$. This idea is justified by the following algebraic manipulation.

If we repeatedly apply (11) to substitute $\hat{d}_i$ by the right-hand side of (11), $\hat{d}_i[n_i^k]$ can be rewritten as

$$\begin{aligned} \hat{d}_i[n_i^k] &= (1 - \gamma)\Big(\left\|x[n_i^k] - w_i[n_i^{k-1}]\right\|^2 \\ &\quad + \gamma \left\|x[n_i^{k-1}] - w_i[n_i^{k-2}]\right\|^2 + \cdots + \gamma^{k-1} \left\|x[n_i^1] - w_i[0]\right\|^2\Big) \\ &= (1 - \gamma) \left(\sum_{j=0}^{k-1} \gamma^j \left\|x[n_i^{k-j}] - w_i[n_i^{k-j-1}]\right\|^2 \right) \\ &= (1 - \gamma^k) \frac{\sum_{j=0}^{k-1} \gamma^j \left\|x[n_i^{k-j}] - w_i[n_i^{k-j-1}]\right\|^2}{\sum_{j=0}^{k-1} \gamma^j}. \end{aligned} \tag{12}$$

Notice in (12), $\gamma^k \rightarrow 0$ for large k, hence $\hat{d}_i$ represents a weighted average of the distortion for neuron i. The

Exhibit BB Page 518

weights $\{\gamma^j\}$ are set up such that more recent distortion measures are emphasized in the averaging process. Notice also, by adjusting the value of γ, we can control the *temporal window size* in the averaging process. The closer γ is to 1, the wider the effective window size; hence, more samples are taken into the averaging process.

Similar to (11), we define the operational measure of P_i as

$$\hat{P}_i[n] = \gamma \hat{P}_i\,[n-1] + (1-\gamma)\,In(x[n], V_i[n-1]) \tag{13}$$

where

$$In(x[n], V_i[n-1]) = \begin{cases} 1 & \text{if } x[n] \text{ in } V_i[n-1] \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

Every time a neuron i is selected, $\hat{d}_i$ and $\hat{P}_i$ are checked to see if (10) is satisfied. If so, then a new neuron will be generated.

Now comes the second question: where to put the newly generated neuron in the lattice?

The procedure we use to place the new neuron is as follows.

Step 1: Find *acceptable empty lattice sites* within the neighborhood region of the parent neuron and list them in order of preference.

Step 2: If the list generated in Step 1 is not empty, place the new neuron on the position specified by the top entry of the list.

Step 3: If the list is empty, *move the lattice* toward the *desired direction* to make room for the new neuron (this operation is called *lattice expansion*, see Fig. 2). Go to Step 1.

Since we want to preserve the structure of the input pattern space on the network, when generating new neurons we must be careful about selecting lattice sites. The criterion for selecting a new site is that it must be at a position where better representation power is needed. This criterion can be evaluated based on the distribution of distortion on different *local lattice directions*. For each neuron i, we can define the local axes by looking into the context of neuron input weight vectors in the neighborhood of neuron i. For example, if neuron j is the positive x direction neighbor of neuron i on the lattice, then we can define the local $+x$ direction for neuron i as

$$\mathbf{x}_i^+ = \Lambda(\mathbf{w}_j - \mathbf{w}_i) \tag{15}$$

where $\Lambda(\,)$ is the vector normalization operator

$$\Lambda(y) \overset{\text{def}}{=} \frac{y}{\|y\|}. \tag{16}$$

With the local axes defined for a given neuron, we can then define the distribution of average distortion on different local axes. For example, the average distortion on

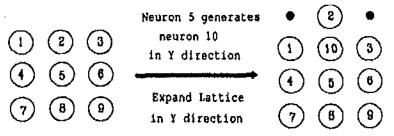


Fig. 2. Lattice expansion. In this example, neuron 5 wants to generate a new neuron in the $+y$ direction and, since there is no empty site within the neighborhood of neuron 5, the lattice is expanded toward the $+y$ direction to accommodate the newly generated neuron 10.

the $+x$ axis for neuron i can be defined as

$$\hat{d}_i\,.\,x^+[n_i^k] = \gamma \hat{d}_i\,.\,x^+[n_i^{k-1}] + (1-\gamma)\Big(r\big((\mathbf{x}[n_i^k] - \mathbf{w}_i[n_i^{k-1}])\,.\,\mathbf{x}_i^+\big)\Big)^2 \tag{17}$$

where $r(\,)$ is the unit ramp function[3] $\hat{d}_i\,.\,x^-$, $\hat{d}_i\,.\,y^+$, $\hat{d}_i\,.\,y^-$ can be defined likewise.

Sometimes the local axis along a given direction can be inferred from other lattice directions, even if the neighbors along that direction are missing. For example, if neuron i does not have $-x$ direction neighbors, but it has $+x$ direction neighbors, then $\mathbf{x}_i^- \leftarrow -\mathbf{x}_i^+$ by inference.

Each neuron also keeps track of a measure called *alias energy*, which is the average distortion along the axes perpendicular to all of the current axes. The operational definition of the alias energy of neuron i is as follows:

$$\hat{d}_i^\perp[n_i^k] = \gamma \hat{d}_i^\perp[n_i^{k-1}] + (1-\gamma) \cdot \left\|\mathbf{Proj}^\perp\big((\mathbf{x}[n_i^k] - \mathbf{w}_i[n_i^{k-1}]), i\big)\right\|^2 \tag{18}$$

where

$$\mathbf{Proj}^\perp(y, i) \overset{\text{def}}{=} y - \sum_j (y\,.\,\mathbf{a}_i^j)\mathbf{a}_i^j \tag{19}$$

is called the *alias operator*, which maps the vector y (in this case $\mathbf{x} - \mathbf{w}_i$) to its alias component related to all the local axes $\{\mathbf{a}_i^j\}$ of neuron i. $\hat{d}_i^\perp$ is used as a measure to determine whether current axes for neuron i are sufficient to represent the input patterns. If $\hat{d}_i^\perp$ is high, a new axis needs to be generated. When a new axis is generated for neuron i, the new direction $\mathbf{a}_i^\perp$ is set to be $\Lambda(\mathbf{Proj}^\perp((\mathbf{x} - \mathbf{w}_i), i))$.

Whenever a new neuron is generated, its input weight vector is set to be

$$\mathbf{w}_{\text{new}} = \mathbf{w}_i + \delta\sqrt{\hat{d}_i\,.\,a^j}\,\mathbf{a}_i^j \tag{20}$$

where $\mathbf{a}_i^j$, the a^j axis for neuron i, is the local direction to put the new neuron; $\hat{d}_i\,.\,a^j$ is the average distortion along axis a^j for neuron i; δ, a number between 0 and 1, is used to control the similarity between the newborn neuron and its parent.

[3]A unit ramp function $r(x)$ is defined to be $r(x) = x$ if $x \geq 0$; $= 0$ otherwise.

Exhibit BB Page 519

2) Neuron Annihilation and Coalition Process: In two circumstances, the number of neurons in SPAN need to be decreased.

- A neuron is not *active* for a long period of time.
- A group of neurons are very similar to one another, such that all of them together overrepresent the patterns in their Voronoi regions.

In the first case, the *virtually dead* neuron can be killed because it is not contributing to the overall system output. An example of this case is when no input pattern falls into the Voronoi region of a neuron. The process of killing this neuron is called *neuron annihilation*.

To determine whether a neuron is active or not, we define a state variable called *activity measure* for neurons, which is an operational definition of the average output activity of a neuron.

$$\hat{Act}_i[n] = \gamma \hat{Act}_i[n-1] + (1-\gamma) y_i[n] \quad (21)$$

where y_i is the output level for neuron i, in SPAN $y_i[n]$ is emulated by $\phi(\| C(x[n]) - l_i \|)$, where $C(x[n])$ is the lattice position of the neuron which is the closest neighbor to the source signal $x[n]$.

If after a long training period, $\hat{Act}$ is very low for a particular neuron, then that neuron should be deleted. Similar to (10), neuron i will be deleted if

$$\hat{Act}_i < \frac{\epsilon_a}{M}. \quad (22)$$

For the second case, if a set of neurons are very similar to each other, then the group of neurons need to be merged to form a less populated group with the members in the new group inheriting attributes from the members in the old one. The merge process can be performed pairwise; that is, check the similarity between neighboring neuron pairs periodically, if the input weight vectors of them are too close, one of the neurons can be eliminated and the remaining one will have the weight vector set to be the average of the two neurons. This process is called *neuron coalition*.

In both neuron annihilation and coalition, one neuron needs to be removed from the network. Before removing this neuron, a criterion needs to be satisfied, that is, removing the neuron should not cause the graph $\{\mathcal{S}, \mathfrak{N}\}$ to become disconnected (where $\mathcal{S}$ is the set of neurons and $\mathfrak{N}$ is the set of neighborhood for the network). This criterion is a necessary condition for the network to retain self-organizing capability.

If deleting a neuron would empty a whole column or row in the lattice, then that column or row can be deleted—this operation is called *lattice shrinkage* (see Fig. 3). The purpose of lattice shrinkage is to keep the representation simple and to retain as much local interaction between neurons as possible.[4]

[4]The self-organization ability of the network increases with the degree of local interaction between neurons. Lattice shrinkage may cause original noninteractive neurons to become neighbors, hence, increasing the degree of interactions in the network.

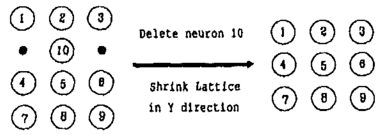
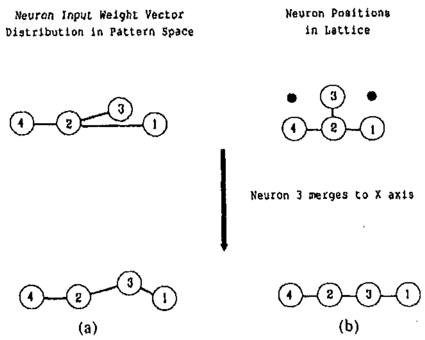


Fig. 3. Lattice shrinkage. Neuron 10 is the last element in a row. After neuron 10 is killed, the whole row becomes empty. The row is then deleted to keep the representation minimium.


Neuron Input Weight Vector
Distribution in Pattern Space
Neuron Positions
in Lattice
Neuron 3 merges to X axis
(a)
(b)
Fig. 4. Axis merging process. (a) Neuron 1 and neuron 3 are the $+x$ and $+y$ direction neighbors of neuron 2, respectively. As the local $+y$ axis (defined by the relative weight vector between neuron 3 and neuron 2) approaches the $+x$ axis (defined by the relative weight vector between neuron 1 and 2), the local dimension of the signal source distribution around neuron 2 decreases from 2 to 1; (b) neuron 3 is merged to the x axis to reflect the change in local dimension.

3) Axes Merge Process: As the statistics of the source signal evolve, the local dimension of a neuron might change. For example, as shown in Fig. 4(a), within the context of the neighborhood of neuron 2, the orientation of the $+y$ axis is very close to that of the $+x$ axis. If this happens, one of the neurons (in this case, neuron 3) should be moved to the local axis that the other neuron resides in (in this case, $+x$ axis), as depicted in Fig. 4(b). This process is called *axes merge*. Notice that after the axes merge, the local dimension of neuron 2 decreases from 2 to 1.

The criterion for deciding when to make axes merge for neuron i is to check the directional cosine between the relative input weight vectors from neuron i to the neighboring neurons. For example, if neuron j is a neighbor on the $+x$ axis and neuron k is a neighbor on the $+y$ axis of neuron i, then for every certain period of time, check

$$\Theta^{+x,+y} = \cos^{-1}\left(\Lambda(w_j - w_i) \cdot \Lambda(w_k - w_i)\right) \quad (23)$$

if the value is too small, then one of the neighbors should be moved to the other axis.

In general, the criterion governing the axes merge process of neuron i is: for every pair of local axes a^{α}, a^{β} on different dimensions,[5] if there exist closest neighbors, neurons j and k, in both directions, then check

$$\Theta_i^{a^\alpha, a^\beta} = \cos^{-1}\left(\Lambda(w_j - w_i) \cdot \Lambda(w_k - w_i)\right) \quad (24)$$

[5]Not all local axes are on different dimensions, e.g., $+x$ and $-x$ are on the same dimension.

Exhibit BB Page 520

if

$$\Theta_i^{a^\alpha, a^\beta} < \epsilon_\Theta \tag{25}$$

then one of the neighbors (in this case, neuron j or neuron k) should be moved to the other axis.

4) Other Lattice Structure Modification Processes: We can define other lattice-structure modification operations for SPAN. For example, a neuron could jump to an empty site in its neighborhood if the context within the neighborhood of the new site is *better* than the old one in representing the pattern space. We call this process *neuron migration*. Two neurons can also switch positions if that is better for both of them, and this process is called *neuron swapping*. We can define some structural merit criterion, which is a function of the neuron weight vectors within the neighborhood of a neuron. Then, this merit criterion can be used as an energy function to guide the neuron migration and swapping processes. Some desirable features are: more interaction between neighboring neurons, local axes in good relative orientation (e.g., x and y axes should be perpendicular to each other), etc.

D. Simulation Results

A simulator for SPAN is implemented using doubly linked lists [23]. We will show the network evolution process of a simple test example here to demonstrate the ideas presented so far.

The network in this example is a 2-D SPAN with a MHNS (minimum homogeneous neighborhood system).[6] The input pattern space is also two-dimensional. We show the network evolution process for two test cases.

Case 1: The signal in the x_1 direction (corresponding to the w_1 direction of the neuron input weight vector) is uniformly distributed between 0.0 and 10.0. The signal in the x_2 direction (corresponding to the w_2 direction of the neuron input weight vector) gradually increases in distribution range and is uniformly distributed within that range. The following summarizes the time-varying statistics of the input pattern distribution.

Time Frame	Iteration Range	x_1 Range	x_2 Range
1	1 → 5000	$0.0 \le x_1 \le 10.0$	$5.0 \le x_2 \le 5.0$
2	5001 → 15000	$0.0 \le x_1 \le 10.0$	$4.0 \le x_2 \le 6.0$
3	15001 → 25000	$0.0 \le x_1 \le 10.0$	$3.0 \le x_2 \le 7.0$
4	25001 → 290000	$0.0 \le x_1 \le 10.0$	$0.0 \le x_2 \le 10.0$

We initialize the lattice with one neuron. The neuron-generating threshold ϵ_d is set to be 0.25 in this case.

Fig. 5 shows the distribution of neuron input weight vectors in the source signal space and the lattice structure during the network evolution process. In each graph, the circles represent neurons and the number inside each circle is the neuron ID. Neuron ID's are ordered according to the sequence of generation.

Initially, the source signals are distributed along one dimension (time frame 1), the lattice also only grows along one dimension. Notice also that the neuron sequence in the lattice preserves the order of neuron input weight vectors in the pattern space. Later on, as the signal distribution in the x_2 direction is turning wide, neurons start growing on the second lattice dimension to follow the change in input pattern space. Finally, as the source signal distribution becomes to cover the region $(0, 0) \le (x_1, x_2) \le (10, 10)$, the network structure also grows to become a perfect rectangle lattice to cover the whole signal distribution range.

Case 2: The network is initialized with the final configuration of Case 1, i.e., 100 neurons arranged in a 10 × 10 lattice with their weight vectors distributed evenly in the range $(0, 0) \le (w_1, w_2) \le (10, 10)$.

The time-varying statistics of the signal source follow the reverse path as Case 1, that is, x_1 distribution spans through the whole range and x_2 distribution shrinks to zero size. The following summarizes the time-varying statistics of the source signals.

Frame No.	Iteration Range	x_1 Range	x_2 Range
1	1 → 5000	$0.0 \le x_1 \le 10.0$	$1.0 \le x_2 \le 9.0$
2	50001 → 250000	$0.0 \le x_1 \le 10.0$	$5.0 \le x_2 \le 5.0$

Fig. 6 shows the network reduction process for this case. As we can see from the graphs, redundant neurons are killed and axes are merged, and finally the lattice is reduced to a one-dimensional structure as the source signal space becomes one-dimensional. We use $\epsilon_a = 0.2$ and $\epsilon_\Theta = \pi/6$ for this case.

III. Coding Scheme

Based on the simulation results presented in the previous section, we know that SPAN has the following features.

- The spatial context of the input pattern space is preserved on the local structure of the lattice. That is, the structural relationships between neurons capture the local structure of the source signal distribution.
- If the structure of the input pattern space changes with time, the network can adapt its structure to follow the change.
- The network adaptation process is incremental and is done through local interaction between neurons; hence, no global structural information is required.

For most kinds of physical sources, the adjacent signals in a series tend to be highly correlated. For example, adjacent pixel blocks are similar in images, and neighboring frames of LPC parameters in speech representation tend to be alike. This is because most of the physical signal-generating mechanisms can only change gradually and continuously. This phenomenon transforms into the representation of a vector quantizer, which shows that adjacent source signals tend to fall into the Voronoi regions that are close in the pattern space.

The above observation suggests that we can utilize the features of SPAN in two phases of the coding process.

[6] A minimum homogeneous neighborhood system (MHNS) is an HNS with $r = 1$ (see Fig. 1).

Exhibit BB Page 521

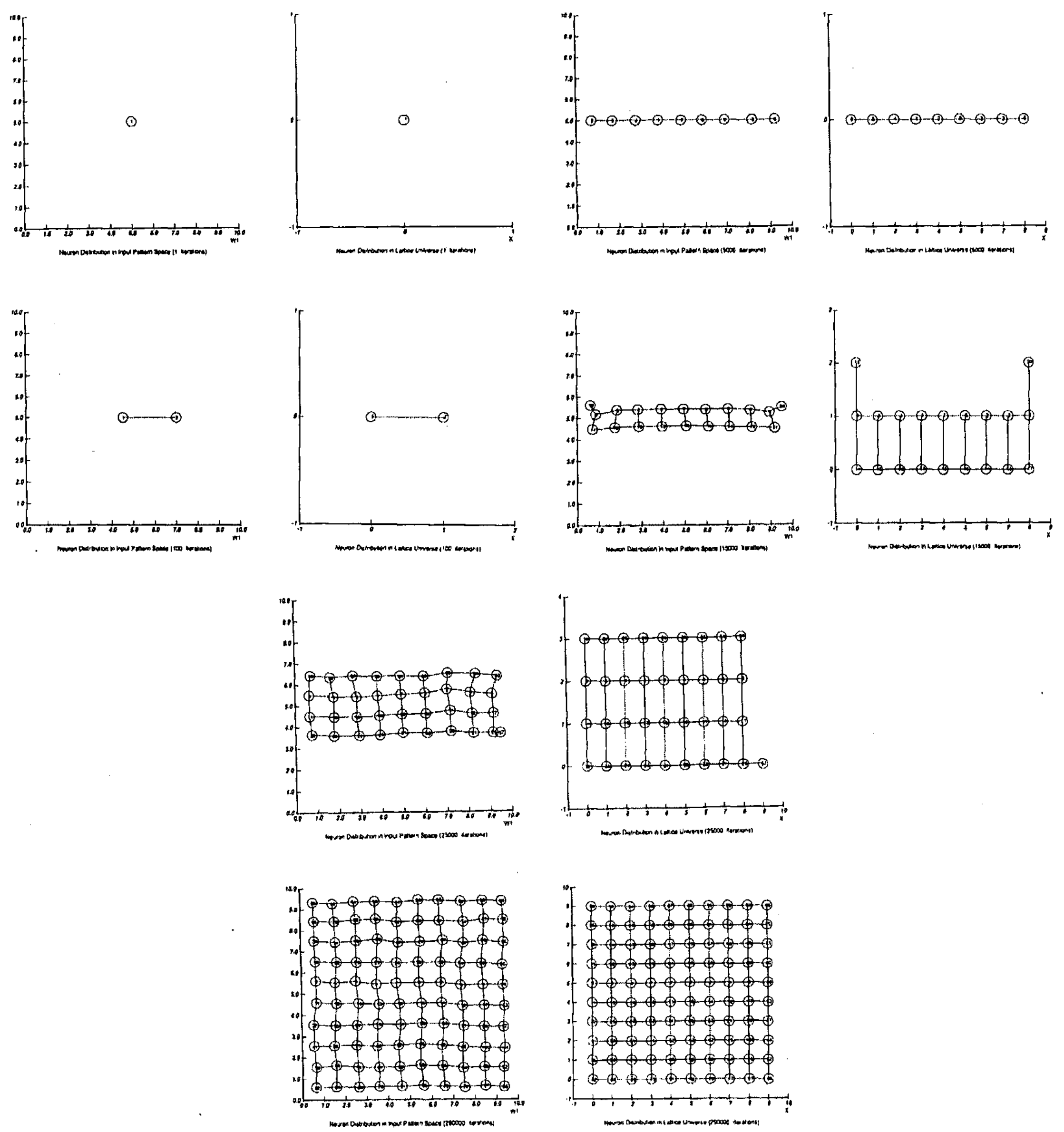


Fig. 5. An example showing network growing process. Each graph on the left column displays the distribution of neuron weight vectors in the source signal space and each graph on the right column shows the lattice structure of the network. The circles in each graph represent neurons and the number inside each circle is the neuron ID, which is ordered in the sequence of generation. Each pair of neighboring neurons is connected by a bar. We have a MHNS as this example.

• *Codebook Search Phase:* The local context within neuron neighborhoods can be used to guide the codebook search process.

• *Encoding Phase:* If we encode only the lattice displacement vector between neurons representing adjacent source signals and the coding scheme is in such a way that shorter lattice displacement vector requires fewer bits, then the bit rate for the whole sequence of source vectors might be minimized. Following this idea, we develop a coding scheme, called the *path coding method*, for SPAN.

Exhibit BB Page 522

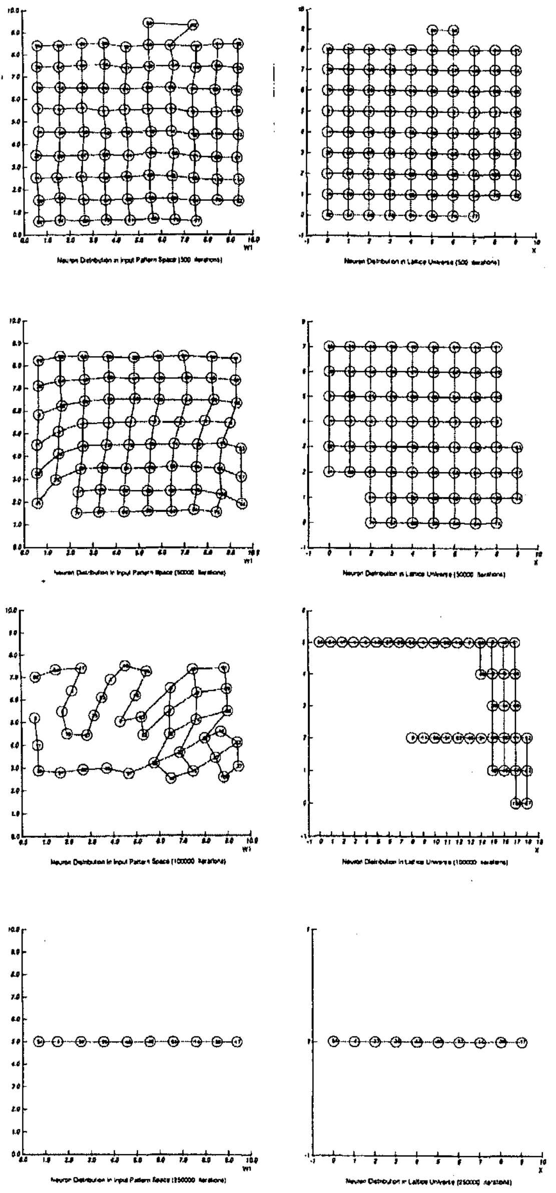


Fig. 6. An example showing the network reduction process. See the caption of Fig. 5 for the meaning of symbols.

A. Fast Codebook Search Procedure

Suppose l_{old} is the lattice position of the previously selected neuron. If the current source signal is x, then the codebook search procedure is as follows:

```
begin
    $l_{\text{try}} \leftarrow l_{\text{old}}$
    while $(\neg(\|x - y(l_{\text{try}})\| \leq \|x - y(l_k)\|, \forall k \in \mathfrak{N}(l_{\text{try}})))$
        $\Delta l \leftarrow$ the lattice index of the point in $\mathcal{L}(l_{\text{try}})$ closest to $(x - y(l_{\text{try}}))$
        $l_{\text{try}} \leftarrow l_{\text{try}} + \Delta l$
    end
    $l_{\text{new}} \leftarrow l_{\text{try}}$
end
```

where $y(l_{\text{try}})$ is the codeword for the neuron at lattice position l_{try}; $\mathfrak{N}(l_{\text{try}})$ is the neighborhood of the neuron at l_{try}; l_k is lattice index for neuron k; $\mathcal{L}(l_{\text{try}})$ is the lattice spanned by the local axes of neuron l_{try},[7] and l_{new} is the lattice index for the newly selected neuron (i.e., the neuron to represent x).

Basically, what the above procedure does is to repeatedly use local axes of neurons to find the next trial position on the lattice until the closest neighbor to the source signal is found.

If the codewords are addressed by their indexes in the lattice, then the number of table access in the search process depends only on how accurately the difference between the source signal and the previous codeword can be represented by the local lattice within the neighborhood of the previously selected neuron. For highly correlated signal sources, the search complexity tends to be of the order $O(1)$ on the number of codewords in the codebook. This is better than the search complexity $O(\log(n))$ for tree-structured codebooks, and the complexity $O(n)$ for full search codebooks.

B. Path Coding Method

The path coding method can be considered a DPCM on the lattice or a vector version of delta modulation, where only the information about lattice displacements between adjacent selected codewords is transmitted through the channel. To represent the lattice displacement, we encode only the transition from each neuron to the neurons in its neighborhood. If a lattice displacement vector is beyond the neighborhood region, then it is represented by the concatenation of transitions. The series of transitions needed to represent a lattice displacement vector from one neuron position to the other neuron position forms a path between the two neurons.

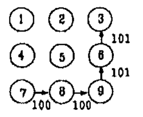
A possible coding scheme for the transitions in 2-D MHNS (minimum homogeneous neighborhood system) is shown in the following.

Transition	Code
$(0, 0) \to (0, 0)$	0
$(0, 0) \to (1, 0)$	100
$(0, 0) \to (0, 1)$	101
$(0, 0) \to (-1, 0)$	110
$(0, 0) \to (0, -1)$	111

The code for a path is the concatenation of transition codes. Fig. 7 shows an example of path code.

[7] At any moment, each neuron can be uniquely designated by its position in the lattice, hence, we may just use the lattice position to denote a neuron.

Exhibit BB Page 523



Code for the path from 7 to 3 is 100100101101

Fig. 7. A path code example.

To make the sequence of path codes *uniquely decodable*, we need to specify the end of each path if the path code is not 0. This can be done by adding a 0 at the end of each path. By coding in this way, a simple finite-state automata can be used to uniquely decode the path sequence.

The following lists some examples of path sequence codes; each pair of adjacent path codes in a sequence is separated by a blank in order to make the codes more understandable to the readers.

Path Sequence	Code
(0, 0) (0, 0) (0, 0) (1, 0) (0, 0)	0 0 0 1000 0
(1, 0) (0, −2)	1000 111110
(0, 0) (2, 0) (0, 1) (0, 0) (0, 0)	0 1001000 101 0 0 0

C. *Performance Comparison*

We compare the performance of the SPAN coder to the LBG coder on 2-D Gauss-Markov (GM) sources defined by the difference equation

$$x[n] = ax[n-1] + g[n] \tag{26}$$

where a is the autoregression constant and $\{g[n]\}$ is a sequence of zero-mean, independent, identically distributed Gaussian random variables. We consider here only the highly correlated case of $a = 1$.[8] The similarity between adjacent source vectors is then controlled by the variance σ^2 of $\{g[n]\}$. The signal source x is two-dimensional and distributed within the region of $(0, 0) \leq (x_1, x_2) \leq (10, 10)$, where x_1 and x_2 are the first and second components of x, respectively.

We simulate the operation of LBG and SPAN with the source model defined above for different σ's. Tables I and II are the simulation results showing the coding performance for an LBG coder and SPAN coder using the path coding method, respectively. For an LBG coder, the distortion and rate are essentially insensitive to σ; hence, for every codeword number, only one distortion is listed in Table I.

On the other hand, from Table II we found that the rate decreases with σ; hence, with the path coding method, the smaller the σ, the better the performance of SPAN.

Fig. 8 shows the distortion rate curves generated by the data from Tables I and II. By comparing the distortion rate curves for LBG and SPAN coders, several interesting points are observed.

[8]In such a case, $\{x[n]\}$ is actually reduced to a Wiener process.

TABLE I
SIMULATION RESULT OF LBG CODER ON 2-D GAUSS-MARKOV SOURCES. THE SOURCE SIGNALS ARE DISTRIBUTED UNIFORMLY IN THE REGION $(0, 0) \leq (x_1, x_2) \leq (10, 10)$

No. codewords	Rate (in bits)	Distortion
1	0	15.23827
2	1	9.993180
4	2	3.805799
8	3	2.169564
16	4	1.020008
32	5	0.5383006
64	6	0.2760772
128	7	0.1385902

TABLE II
SIMULATION RESULTS OF SPAN CODER ON GAUSS-MARKOV SOURCES USING THE PATH CODING METHOD. σ IS THE STANDARD DEVIATION OF THE GAUSSIAN COMPONENT IN A GAUSS-MARKOV SOURCE; THE AVERAGE PATH LENGTH IS THE AVERAGE NUMBER OF NEIGHBORHOOD TRANSITIONS REQUIRED TO SPECIFY THE DISPLACEMENT VECTOR BETWEEN ADJACENT SOURCE VECTORS IN THE TRAINING SEQUENCE; THE RANGE OF SOURCE SIGNAL DISTRIBUTION IS $(0, 0) \leq (x_1, x_2) \leq (10, 10)$

σ	No. Codewords	Lattice Size	Average Path Length	Average Rate (in bits)	Distortion
1.0	6	3 × 2	0.26275001	1.788233	2.852761
	9	3 × 3	0.3443333	2.032989	1.774432
	12	4 × 3	0.4365917	2.309767	1.383733
	25	5 × 5	0.6898200	3.069456	0.6541284
	48	8 × 6	1.017548	4.052642	0.3557738
	100	10 × 10	1.496417	5.489250	0.1655245
	132	12 × 11	1.731668	6.195004	0.1259496
0.5	6	3 × 2	0.1338500	1.401533	2.998598
	9	3 × 3	0.1670444	1.501122	1.809933
	12	4 × 3	0.2109750	1.632917	1.396291
	25	5 × 5	0.3325640	1.997688	0.6550556
	48	8 × 6	0.5022479	2.506742	0.3548502
	100	10 × 10	0.7475840	3.242751	0.1648479
	132	12 × 11	0.8727659	3.618297	0.1256384
0.2	6	3 × 2	5.61×10^{-7}	1.168283	3.014576
	9	3 × 3	6.227778×10^{-2}	1.186822	1.874519
	12	4 × 3	8.1325×10^{-2}	1.243967	1.449642
	25	5 × 5	1.3254×10^{-1}	1.397616	0.6743547
	48	8 × 6	1.982792×10^{-1}	1.594835	0.3606514
	100	10 × 10	2.94362×10^{-1}	1.883085	0.1653093
	132	12 × 11	3.422167×10^{-1}	2.026649	0.1259553

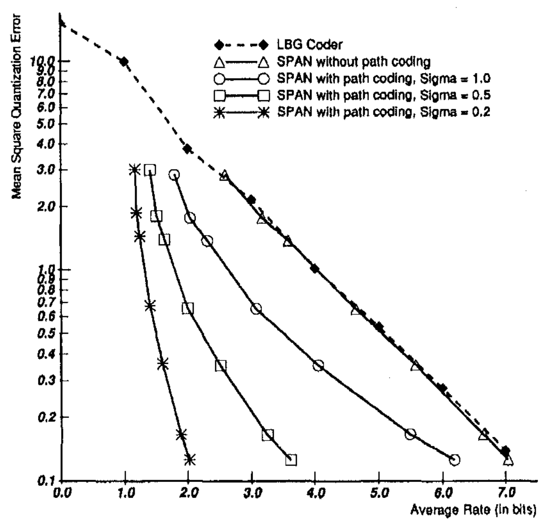


Fig. 8. Performance comparison between LBG and SPAN coder on Gauss-Markov sources.

• For SPAN not using the path coding method (fixed rate), the performance is similar to LBG.

• For SPAN using the path coding method, the perfor-

Exhibit BB Page 524

mance is much better than LBG for highly correlated sources.

- The higher the degree of similarity between adjacent source vectors, the better the performance of SPAN is by using the path coding method.
- As σ decreases, the rate becomes less sensitive to the size of the codebook. This means that the rate *does not scale up* with the codebook when the signal source is highly correlated.

In general, by using the path coding method, the bit rate only depends on the correlation between source signals and is quite insensitive to the size of the codebook. Extreme benefits can be gained by using this method when source signals are highly correlated.

However, if the source signals are totally uncorrelated, the path coding method might generate a worse performance than the LBG. When this situation occurs, the system should switch back to the normal encoding scheme but can still use a SPAN codebook and performance as good as LBG can be retained. This switching process should not be too difficult to implement; we must only check the *average path length* between adjacent selected codewords in SPAN periodically. If the average path is too long, then switch back to the normal coding scheme. Of course, the same decision criterion should be used on both sides of the channel.

Fig. 9 shows the codebook searching performance using our proposed fast codebook search procedure. Notice that the codebook searching time is quite insensitive to the size of the codebook.

IV. Adaptive SPAN Coder/Decoder

Since the adaptation process of SPAN is incremental and requires only local operation on the lattice, we can use this property to design an adaptive vector quantizer. Fig. 10 shows a proposed adaptive source coding system based on the SPAN codebook.

As shown in Fig. 10, at time index n, the input source vector $x[n]$ is fed into the coder. The neuron in SPAN with an input weight vector closest to $x[n]$ is then selected. If we let $l[n]$ be the lattice position of the selected neuron and $y(l[n])$ be the codeword (i.e., the input weight vector) for neuron $l[n]$. $d[n]$, the lattice path from $l[n-1]$ to $l[n]$ is then encoded using the path coding method to generate path code $c[n]$, where $l[n-1]$ is the lattice position of the previous selected neuron. $c[n]$ is then sent through the *main channel*.

To enable the adaptation process on both sides of the channel, the residue vector $e[n] = x[n] - y(l[n])$ is encoded and the code $r[n]$ representing the residue codeword $\hat{e}[n]$ is sent through the *side channel*. The residue codeword $\hat{e}[n]$ is then used to activate the network adaptation process of SPAN on both sides of the channel (equivalently, on both sides of the channel, the SPAN codebooks see the same input signal $\hat{x}[n] = y(l[n]) + \hat{e}[n]$, and hence will adapt themselves with the same pace).

The residue encoding is carried out by a small-size *lattice quantizer* in which no codebook is needed.[9] We can encode the lattice positions in the residue space using the path coding method (encode the difference between the lattice positions and the origin).

Initially, we can have codebooks on both sides of the

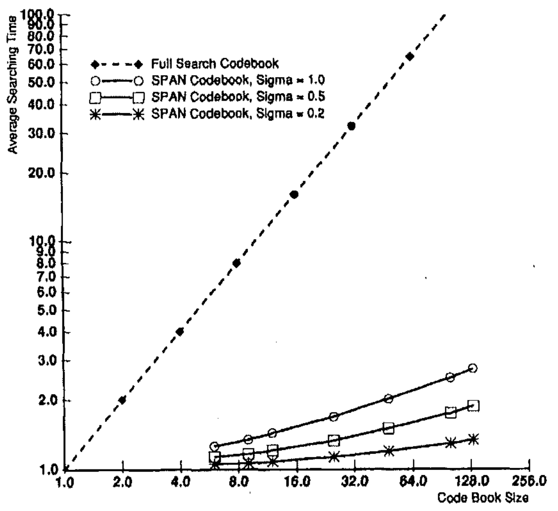


Fig. 9. Codebook searching efficiency of SPAN coder using fast codebook search procedure.

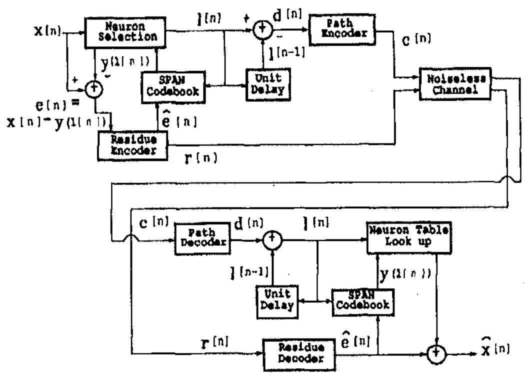


Fig. 10. A proposed adaptive SPAN coding/decoding system. $x[n]$ is the source vector at time n; $l[n]$ is the lattice position of the neuron with input weight vector closest to $x[n]$; $y(l[n])$ is the input weight vector for the neuron location at $l[n]$ in the lattice; $d[n]$ is the lattice displacement vector from the lattice position of the previously selected neuron to that of the current neuron; $c[n]$ is the path code for $d[n]$; $e[n]$ is the vector difference between $x[n]$ and $y([n])$; $\hat{e}[n]$ is the quantized version of $e[n]$ through the residue encoder; $r[n]$ is the code for $\hat{e}[n]$; and $\hat{x}[n]$ is the signal regenerated at the receiver end.

[9] In the lattice quantizer, the space of interests is decomposed regularly by a lattice structure [6]. Every codeword in the quantizer is a lattice point in the source signal space. Since lattice points in the source signal space can be derived through some simple calculation, no codebook storage is necessary. Reference [7] has developed a series of coding/decoding algorithms for lattice quantizers.

Exhibit BB Page 525

channel with the same initial random condition (say both have one neuron residing at the origin). Then, as the source signals come in, the codebooks will evolve with the same pace to follow the signal statistics on both sides of the channel. In the beginning, most of the information is sent through the side channel; then gradually, as the structure of the codebooks is built up and approaches that of the source signal space, more and more information is sent through the main channel. Eventually, most of the information will be transmitted through the main channel, and the side channel will be *shut off*. Later on, if the statistics of the source signal change, the side channel will *wake up* and the adaptation cycle will start again.

Better performance can be obtained by further incorporating some predictive coding mechanism into the encoding part of the system. The function of the predictive mechanism is to predict the next codeword position in the lattice, based on previously selected neuron positions. The lattice position difference between the actually selected codeword and the predicted codeword is encoded using the path coding method.

V. Conclusions

As the signal sources become more and more complicated, there is a growing effort to develop codebooks with an internal structure that reflects some aspect of the signal space in order to enhance the codeword search process [4], [5]. The other trend in vector quantizer design is to incorporate feedback and internal states into the encoder to eliminate the correlation between source signals in the sequence [3], [8], [9]. In this case, the codes are sensitive to the context of the source sequence and finite state automata models are used to capture the local context buried in the sequence, so as to eliminate correlations. The coding/decoding system presented in this paper can be viewed as an attempt to cover both aspects mentioned above.

In this paper, we described the basic framework and structure-level adaptation mechanisms for SPAN, and proposed an adaptive source coding/decoding system using SPAN. In summary, the most attractive features of SPAN are that it can map the local context of the source signal space *conformally* onto a lattice structure and can adapt the lattice structure to follow the statistics of the source signals. We have shown that, by utilizing the local context built in the lattice, we can achieve both fast codebook searching (to enhance encoding efficiency) and source signal correlation elimination (to enhance distortion rate performance) under our proposed framework.

Future work can be done in trying the SPAN coder/decoder scheme on such physical sources as image, speech, etc. It is also possible to apply this model to subband coding systems to allocate bits among several channels. In this case, each channel is taken care of by a SPAN; in addition, an association mechanism can be added on top of the group of SPAN's to adjust the threshold parameters (e.g., ϵ_d, ϵ_a, etc.) of the networks in order to optimize the global system performance.

Appendix
Asymptotic Value of w_i

According to Kohonen's learning algorithm, the iteration equation for the weight adjustment of neuron i is

$$\begin{aligned}
w_i[n+1] &= w_i[n] + \alpha y_i(x[n+1]) \\
&\quad \cdot (x[n+1] - w_i[n]) \\
&= \big(1 - \alpha y_i(x[n+1])\big)\, w_i[n] \\
&\quad + \alpha y_i(x[n+1])\, x[n+1] \\
&= \big(1 - \alpha y_i(x[n+1])\big)\Big(\big(1 - \alpha y_i(x[n])\big) \\
&\quad \cdot w_i[n-1] \\
&\quad + \alpha y_i(x[n])\, x[n]\Big) \\
&\quad + \alpha y_i(x[n+1])\, x[n+1] \\
&= \left(\prod_{k=0}^{n} \big(1 - \alpha y_i(x[n+1-k])\big)\right) w_i[0] \\
&\quad + \alpha \sum_{j=1}^{n+1} \left(y_i(x[j]) \prod_{k=j+1}^{n+1}{}^{+} \right. \\
&\quad \left. \cdot \big(1 - \alpha y_i(x[k])\big)\right) x[j] \qquad (27)
\end{aligned}$$

where

$$\begin{aligned}
\prod_{k=i}^{j}{}^{+} &= \prod_{k=i}^{j} \qquad \text{if } j \geq i \\
&= 0 \qquad \text{otherwise.} \qquad (28)
\end{aligned}$$

In (27),

$$\lim_{n\to\infty} \left(\prod_{k=0}^{n} \big(1 - \alpha y_i(x[n+1-k])\big)\right) = 0 \qquad (29)$$

hence

$$\begin{aligned}
\hat{w}_i &\overset{\text{def}}{=} \lim_{n\to\infty} w_i[n] \\
&= \lim_{n\to\infty} \alpha \sum_{j=1}^{n} \left(y_i(x[j])\, x[j] \right. \\
&\quad \left. \cdot \prod_{k=j+1}^{n}{}^{+} \big(1 - \alpha y_i(x[k])\big)\right). \qquad (30)
\end{aligned}$$

We know that $y_i(x) \neq 0$ iff $x \in V_j$ for some $j \in \mathfrak{N}_i$, where V_j is the Voronoi region for neuron j and $\mathfrak{N}_i$ is the neighborhood set of neuron i. Let $y_i(x) = Y_{ij}$ when $x \in V_j$ and let

$$\begin{aligned}
\gamma_i(m) &\overset{\text{def}}{=} \prod_{k=n-m+1}^{n} \big(1 - \alpha y_i(x[k])\big) \\
&= \prod_{j} (1 - \alpha Y_{ij})^{m_j}
\end{aligned} \qquad (31)$$

when m_j is the number of occurrences of x in V_j during the period of m samples.

Exhibit BB Page 526

For α sufficiently small, $\gamma_i(m)$ can be approximated by

$$\gamma_i(m) \simeq \prod_j (1 - m_j \alpha Y_{ij}) \simeq 1 - \alpha \sum_j m_j Y_{ij}. \quad (32)$$

We also know $\Sigma_j m_j = m$ and $P(j) \simeq m_j/m$ as m becomes large, where $P(j)$ is the probability mass function of the occurrence of input signal in V_j, i.e., $P(j) = \int_{V_j} p(x)\,dx$. Hence, for large m, following (32), we have

$$\gamma_i(m) \simeq 1 - m\alpha \sum_j P(j) Y_{ij} = 1 - \alpha m \chi_i \overset{\text{def}}{=} \hat{\gamma}_i(m) \quad (33)$$

where $\chi_i \overset{\text{def}}{=} \Sigma_j P(j) Y_{ij}$.

For α sufficiently small, $\gamma_i(m) \simeq 1$. Assume $x[n]$ is a stationary process; then $y_i(x[n])$ is also stationary. For large m, we have

$$\begin{aligned}\prod_{k=n-m+1}^{n} &\left(1 - \alpha y_i(x[k])\right) \\ &\simeq \prod_{k=n-2m+1}^{n-m} \left(1 - \alpha y_i(x[k])\right) \\ &\simeq \prod_{k=n-(j+1)m+1}^{n-jm} \left(1 - \alpha y_i(x[k]\right) \\ &\simeq \hat{\gamma}_i(m), \qquad 0 \le j \le \frac{n}{m} - 1. \quad (34)\end{aligned}$$

Using the approximation above, if we partition the summation in (30) into n/m smaller summations with m terms in each summation, we have

$$\begin{aligned}\hat{w}_i \simeq \lim_{n\to\infty} \alpha \Bigg(&\sum_{j=n-m+1}^{n} y_i(x[j])\,x[j] \\ &+ \hat{\gamma}_i(m) \sum_{j=n-2m+1}^{n-m} y_i(x[j])\,x[j] + \cdots \\ &+ \left(\hat{\gamma}_i(m)\right)^{n/m-1} \sum_{j=1}^{m} y_i(x[j])\,x[j] \Bigg). \quad (35)\end{aligned}$$

Let

$$\begin{aligned}\bar{q}_i = E\left(y_i(x)x\right) &= \sum_k P(k)\, Y_{ik} E(x \mid x \in V_k) \\ &= \sum_k P(k)\, Y_{ik} X_k \quad (36)\end{aligned}$$

where X_k is the centroid of Voronoi region V_k.

Then, (35) can be simplified to

$$\begin{aligned}\hat{w}_i &\simeq \alpha \lim_{n\to\infty} \left(m\bar{q}_i \sum_{k=0}^{n/m-1} \gamma_i(m)^k \right) \\ &= \frac{\alpha m \bar{q}_i}{1 - \gamma_i(m)} = \frac{\alpha m \bar{q}_i}{\alpha m \chi_i} = \frac{\bar{q}_i}{\chi_i} \\ &= \frac{\sum_k P(k)\, Y_{ik} X_k}{\sum_k P(k)\, Y_{ik}} = \frac{\int_V p(x)\, y_i(x)\, x\,dx}{\int_V p(x)\, y_i(x)\,dx} \\ &= \frac{\int_{\Omega_i} p(x)\, y_i(x)\, x\,dx}{\int_{\Omega_i} p(x)\, y_i(x)\,dx}. \quad (37)\end{aligned}$$

Notice that, in the column integration above, we replaced the integration range of V by Ω_i. The reason we can do that is because $y_i(x)$ is nonzero only in Ω_i. Thus, the proof.

Acknowledgment

The authors would like to thank the anonymous referees for their comments, which improved the presentation of this work.

References

[1] S. C. Ahalt, P. Chen, and A. K. Krishnamurthy, "Performance analysis of two image vector quantization techniques," in *Proc. IJCNN: Int. Joint Conf. Neural Networks*, June 1989, pp. I-16.

[2] T. Berger, *Rate Distortion Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1971.

[3] P.-C. Chang, "Predictive, hierarchical, and transform vector quantization for speech coding," Ph.D. dissertation, Stanford Univ., Stanford, CA, May 1986.

[4] P. A. Chou, T. Lookabaugh, and R. M. Gray, "Optimal pruning with applications to tree-structured source coding and modeling," *IEEE Trans. Inform. Theory*, vol. 35, no. 2, pp. 299-315, Mar. 1989.

[5] P. A. Chou, "Application of information theory to pattern recognition and the design of decision trees and trellises," Ph.D. dissertation, Stanford Univ., Stanford, CA, June 1988.

[6] J. H. Conway and N. J. A. Sloane, "Voronoi regions of lattices, second moments of polytopes, and quantization," *IEEE Trans. Inform. Theory*, vol. IT-28, no. 2, pp. 211-226, Mar. 1982.

[7] ——, "Fast quantizing and decoding algorithms for lattice quantizers and codes," *IEEE Trans. Inform. Theory*, vol. IT-28, no. 2, pp. 227-232, Mar. 1982.

[8] M. Ostendorf Dunham, "Finite-state vector quantization for low rate speed coding," Ph.D. dissertation, Stanford Univ., Stanford, CA, Feb. 1985.

[9] J. Foster, "Finite-state-vector quantization for waveform coding," Ph.D. dissertation, Stanford Univ., Stanford, CA, Nov. 1982.

[10] R. G. Gallager, *Information Theory and Reliable Communication*. New York: Wiley, 1968.

[11] A. Gersho and V. Cuperman, "Vector quantization: A pattern-matching technique for speech coding," *IEEE Commun. Mag.*, pp. 15-21, Dec. 1983.

[12] A. Gersho, "On the structure of vector quantizers," *IEEE Trans. Inform. Theory*, vol. IT-28, no. 2, pp. 157-166, Mar. 1982.

[13] ——, "Asymptotically optimal block quantization," *IEEE Trans. Inform. Theory*, vol. IT-25, no. 4, pp. 373-380, July 1979.

[14] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Norwell, MA: Kluwer, 1990.

[15] R. M. Gray, "Vector quantization," *IEEE ASSP Mag.*, pp. 4-29, Apr. 1984.

[16] ——, *Source Coding Theory*. Norwell, MA: Kluwer, 1990.

[17] R. M. Gray and L. D. Davisson, *Random Processes*. Englewood Cliffs, NJ: Prentice-Hall, 1986.

[18] C. Kittel, *Introduction to Solid State Physics*, 6th ed. New York: Wiley, 1986, ch. 1.

[19] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biol. Cybern.*, vol. 43, pp. 59-69, 1982.

[20] ——, *Self Organization and Associative Memory*, ch. 5, pp. 119-155.

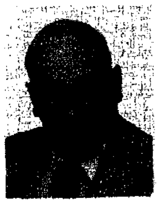
Exhibit BB Page 527

[21] ——, "The neural phonetic typewriter," *Computer*, pp. 11-22, Mar. 1988.
[22] ——, "An introduction to neural computing," *Neural Networks*, vol. 1, no. 1, pp. 3-16, 1988.
[23] T.-C. Lee and A. M. Peterson, "Implementing a self-development neural network using doubly linked lists," in *Proc. COMPSAC89, IEEE 13th Int. Comp. Software Appl. Conf.*, Sept. 20-22, 1989.
[24] T.-C. Lee, "Structure level adaptation for artificial neural networks," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1990.
[25] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. COM-28, pp. 84-95, Jan. 1980.
[26] S. P. Lloyd, "Least square quantization in PCM," *Bell Lab. Tech. Notes*, 1957, also in *IEEE Trans. Inform. Theory*, vol. IT-28, no. 2, pp. 129-137, Mar. 1982.
[27] N. M. Nasrabadi, "Image coding using vector quantization: A review," *IEEE Trans. Commun.*, vol. 36, pp. 957-971, Aug. 1988.
[28] N. M. Nasrabadi and Y. Feng, "Vector quantization of images based upon a neural-network clustering algorithm," *SPIE Vol. 1001: Visual Commun. Image Process. '88, Part 1*, pp. 207-213, Nov. 1988.
[29] ——, "Vector quantization of images based upon the Kohonen self-organization feature maps," in *Proc. IEEE Int. Conf. Neural Networks*, July 1988, pp. I-93.
[30] N. Packard and S. Wolfram, "Two-dimensional cellular automata," *J. Stat. Phys.*, vol. 38, p. 901, 1985.
[31] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379-423, 623-656, 1948.
[32] ——, "Coding theorem for a discrete source with a fidelity criterion," *IRE Nat. Con. Record, Part 4*, pp. 142-163, 1959.
[33] Y. Shoham and A. Gersho, "Efficient bit allocation for an arbitrary set of quantizers," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, pp. 1445-1453, Sept. 1988.
[34] V. V. Tolat and A. M. Peterson, "A self-organization neural network for classifying sequences," in *Proc. IJCNN: Int. Joint Conf. Neural Networks*, June 1989, pp. II-561.
[35] A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding*. New York: McGraw-Hill, 1979.
[36] J. Von Neumann, *Theory of Self-Reproducing Automata*, A. W. Burks, Ed. Urbana and London: University of Illinois Press, 1966.
[37] S. Wolfram, "Statistical mechanics of cellular automata," *Rev. Mod. Phys.*, vol. 55, p. 601, 1983.

Tsu-Chang Lee was born in Taipei, Taiwan, Republic of China, on August 8, 1961. He received the B.S. degree in electrical engineering from the National Taiwan University in 1983, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1987 and 1990, respectively.

His current research interests include neural networks, adaptive signal processing, data compression, and hardware implementation of novel signal processing systems.

Dr. Lee is a member of ACM and AAAI.



Allen M. Peterson (M'56-F'62-LF'90) was born in Santa Clara, CA, on May 22, 1922. He received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1952.

Since 1952, he has been at Stanford University where he is Professor of Electrical Engineering. His research interests include digital signal processing, algorithms, architecture and hardware implementation with applications in telecommunication, radar and remote sensing.

Dr. Peterson is a member of the National Academy of Engineering.

Exhibit BB Page 528