Cascaded Vector Quantization by Non-linear PCA Network Layers

Rüdiger W. Brause
J. W. Goethe-University, FB Informatik
D-60054 Frankfurt, Germany
brause@informatik.uni-frankfurt.de

Abstract

In this paper the different mechanism of principal component analysis (PCA) and vector quantization are combined in an architecture of one functional layer which implements vector quantization without using winner-take-all nets.

After introducing cascaded vector quantization, the paper introduces a new network (the binary cascade network) *which is composed of lateral inhibited neurons for PCA. They have bell-shaped activation functions which provide binary cascaded quantization stages.*

It is shown that this architecture is nearly optimal in terms of ressource distribution.

1 Introduction

The information processing in natural and artificial neural systems is often regarded as a process where the incoming information is compressed and encoded. Among others, there are two important techniques to compress data: principal component analysis (PCA), which transforms most of the signal variance to a few number of channels and neglects the ones with the smallest components, and vector quantization (VQ), which devides the input space into classes of events and assign only one protype event to each class.

Classically, both paradigms can be combined into two stages to form the *transform coding* processing algorithm (Habibi and Wintz [7], which is heavily used in modern compression algorithm as for instance in the JPEG or MPEG picture compression specifications.

In this paper, a version of multilayer transform coding is presented combining both approaches in the implementation by a special neural network, called *binary cascade network.*

2 Cascaded vector quantization

Let us present the idea of cascaded vector quantization by first introducing some known results on vector quantization.

2.1 Vector quantization

Vector quantization is one of the most important data reduction methods currently available. Here, the n-dim input space of patterns $\{\mathbf{x}\}$ is devided into several disjunctive set of patterns. Each set can be seen as a *class* of patterns, represented by a certain *class prototype*, the *codebook vector*. Often, as class prototype the centroid is chosen. The set of all codebook vectors is called the *codebook*. In figure 1 the example partition of a 2-dim pattern space by a regular grid is shown.

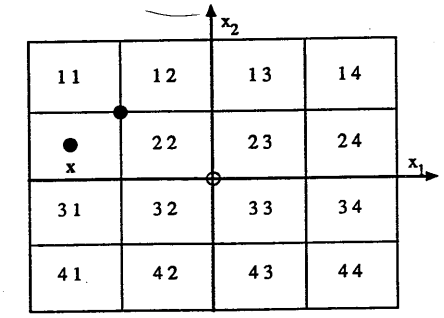

Fig.1 Vector quantization

For the pattern x the according class prototype of class 21 is shown as a black dot.

There are two severe restrictions in the design of vector quantization: the necessary search time for mapping an unknown sample pattern to a codebook vector and the storage amount for storing all the codebook vectors.

The first problem can be avoided if one chooses a regular lattice architecture for the class boarders, as it is shown in figure 1. Here, the class index can be directly calculated in contrast to irregular class boarders where for m classes also m comparisons must be made to detect the proper class cell for a pattern x. Although the lattice form implies class cells with a slightly higher vector quantization error (e.g. 4% for 2-dim squares compared to the optimal cell form of regular hexagons, see Makhoul et al. [11]) it can be shown that lattice quantizer, i.e. quantizer

based on codebook vectors which lie on a regular lattice within the special cell form of parallelohydra (2-dim case: a hexagone), for a fixed number *m* of codebook vectors partition the n-dimensional space in a way which asymptotically for large *m* approaches for any probabability density function (pdf) the performance of the optimal quantizer (Gersho [5]).

If we consider pattern sources where the components are independent, the problem of vector quantization becomes less severe, because we can quantize now all components independently (*scalar vector quantization*) and do not have to pay attention on the cell geometry anymore. For this situation, Gish and Pierce [6] have shown that the quantization interval should be the same in all dimensions, i.e. the class cell form is a n-dim cube. This will result in a difference of 0.255 bits per dimension to the theoretical optimal quantizer in terms of rate-distortion theory, independent of the form of the pdf. Although this is true only for high bit rates, i.e. a high number of codebook vectors, experiments of Farvadin and Modestino [4] showed that the uniform quantizer is also close to optimal at lower bit rates.

2.2 Transform coding

One of the main obstacle for the simple scheme of scalar vector quantization is the assumption of independence of all the pattern components. This is normally not true. Nevertheless, as a good precondition for the independence we can decorrelate the components by a linear transformation. This leaves only the non-linear relations between the components which are often less significant. Thus, the main sequence of this procedure consists of two stages: first a decorrelation step and then the scalar quantization, see figure 2.

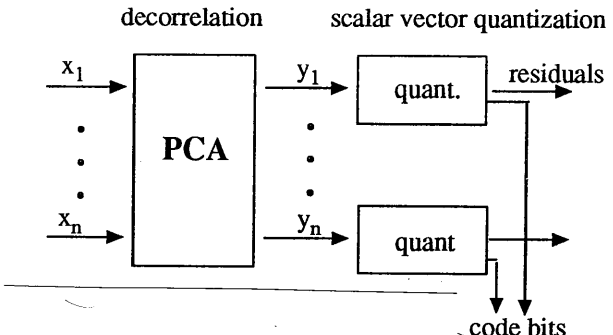

Fig.2 transform coding

For Gaussian sources, Huang and Schultheiss [8] showed that for a given bit rate under all possible transformations which decorrelate the components the linear transformation which minimizes the distortion measure, e.g. the mean squared error, is given by the transformation on the base of the normalized eigenvectors $\mathbf{w}_i$ of the covariance matrix $C=\langle \mathbf{x}\mathbf{x}^T \rangle$ of the centered input $\mathbf{x}$. If the decorrelated components are obtained from Gaussian input, they become also independent.

The eigenvector transformation is also known as *principal component analysis* (PCA) or *Karhunen-Loève transform* (KLT).

The scheme of figure 2 assumes that the number *n* of components remain the same at the two stages. If we reduce instead the number of components to a number m<n, an additional data compression will take place. If we choose as neglected components only those of the eigenvectors with the smallest eigenvalues, the neglected input variance and the resulting distortion will be minimal. This scheme is called *transform coding* and was proposed for picture processing by Habibi and Wintz [7].

2.3 Cascaded vector quantization

The transform coding concept helps to improve the search time (classification time of input pattern $\mathbf{x}$) by a regular vector quantization lattice of codebook vectors. An additional effort must be made to overcome the second mentioned problem: How can we reduce the storage amount for the codebook vectors?

For this purpose, let us reconsider the situation in figure 1. Here, the quantization can also be broken into two steps. Instead of searching directly for the class prototype 21, we first devide the whole area into four superclasses, formed by the class unions {11,12, 21, 22}, {13, 14, 23, 24}, {31, 32, 41, 42}, {33, 34, 43, 44}. The correspondent class prototypes (black dots • in fig.1) are assumed to be in the middle of the classes, i.e. at the common point of the four class boarders. Thus, to classify our example $\mathbf{x}$ in figure 2, we first map it to the class prototype $\mathbf{w}^{(1)}$. Within this superclass, we decide now to which class the pattern should be assigned. The corresponding relative vector of the class prototype we denote by $\mathbf{w}^{(2)}$ and the remaining quantization error we denote by $\mathbf{d}$. The resulting sequential decomposition is shown in figure 3.

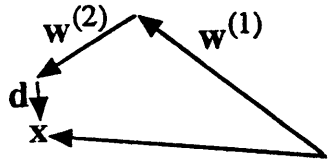

Fig.3 The two-stage decomposition

For b decisions at each stage, this kind of sequential code book search results in a search time of only $\log_b(m)$ steps in contrast to a full search of *m* steps is is known as *tree-searched VQ*.

Now, let us go one step further. If we regard figure 1 closer, we can see that, due to the regularity in the grid, the relative class prototypes $\mathbf{w}_i^{(2)}$ are the same in all the

four superclasses. Without loss of information, we can replace the four sets of prototypes $\{w_i^{(2)}\}$ by just one set and save a lot of storage space. More accurately, instead of using $m=m_1m_2=16$ codebook vectors with $m_1=4$ vectors in the first and $m_2=4$ vectors in the second stage, we only use $m=m_1+m_2=8$ codebook vectors.

Thus, by dividing the vector quantization process into several stages called *cascaded vector quantization*, we can save an important amount of search time and storage space [9]. Since a major item in the scalar quantization process is the independence of the components, a PCA stage before each quantization stage improves the results by decorrelating the components [15].

3 The optimal architectural parameters

In this section we will shortly explore the paradigm of cascaded vector quantization to deduce optimal architectural conditions for an implementation. Since we do not know anything about the pdf of the input patterns, let us evaluate the optimal number of units per layer for the uniform probality distribution. As we can see in section 5, this also covers much of initially differently sensored pdf's, such as Gaussian sources.

3.1 The optimal number of units per layer

Denoting the interval length in each dimension by A, we assume that all input patterns are in the cube with the volume A^n. This is the case when we already had a variance normalization stage before. The mean squared error (or pattern variance σ^2) in the cube is according to appendix A.2

$$\langle d^2\rangle^{(0)} = n\, A^n\, (A^2/12)$$

If we devide the cube A^n into m_1 cubic classes each one with the cube length a, the mean squared error per class is $\langle d^2\rangle = na^n(a^2/12)$ and the whole error of this stage becomes, using $a^n=A^n/m_1$ or $a=A/(m_1^{1/n})$,

$$\langle d^2\rangle^{(1)} = m_1\,(n\,a^n\,(a^2/12)) = n\,A^n\,(A^2/12m_1^{2/n})$$
$$= \langle d^2\rangle^{(0)}\, m_1^{-2/n} = \sigma^2(0)\, f(m_1) \qquad (3.1)$$

Thus, the residual variance $\sigma^2(k)=\langle d^2\rangle^{(k)}$ at each quantization stage k formally can be described by the reduction of the previous variance by a function f(m,k)

$$\langle d^2\rangle^{(k)} = \sigma^2(k) = \sigma^2(k-1)\, f(m,k) \qquad (3.2)$$

This is also true for other quantization schemes. The function f(.) takes its minimum at m=infinity and its maximum of one at m=1.

Now, let us calculate the optimal numbers of units per layer. As restriction, we have constant resources, i.e. a constant number $m=\Sigma_i m_i$ of codebook vectors. In the later used terminology of neural networks, this corresponds to a constant number of neurons to use in the quantization. At the k-th stage, we have with equation (3.2), recursively replacing the variance, the equation

$$\sigma^2(k) = \sigma^2(0)\prod_{i=1}^{k} f(m,i) \qquad (3.3)$$

Let us compute the best distribution of m_i for the different stages with the constraint $m-\Sigma_i m_i=0$ by using the method Lagrange multipliers.

$$L(m_1,...,m_k,\mu) = \sigma^2(0)\prod_k f(m,k) + \mu(m-\Sigma_i m_i)$$

The necessary conditions for a multivariate extremum (here: a minimum) are given by the conditions

$$\partial L/\partial m_i = 0 \quad \forall i\,,\ \partial L/\partial \mu = 0 \qquad (3.4)$$

which means for $\partial L/\partial m_i = 0 = \partial L/\partial m_j$

$$\sigma^2(0)\prod_{k\neq i} f(m,k) + \mu = 0 = \sigma^2(0)\prod_{k\neq j} f(m,k) + \mu$$

or

$$f(m,j)\,\frac{\partial f(m,i)}{\partial m_i} = f(m,i)\,\frac{\partial f(m,j)}{\partial m_j}$$

$$\frac{1}{f(m,i)}\,\frac{\partial f(m,i)}{\partial m_i} = \frac{1}{f(m,j)}\,\frac{\partial f(m,j)}{\partial m_j}$$

and

$$\frac{\partial}{\partial m}\log f(m,i) = \frac{\partial}{\partial m}\log f(m,j)$$

Since f(m) and its logarithm are monotone functions without additive constants, the derivatives are only identical if the arguments are identical, i.e.

$$m_i = m_j \qquad \forall i,j. \qquad (3.5)$$

Thus, we have concluded that every layer should have the same number of classes. This is true for all pdf's which have property (3.2).

3.2 The optimal number of layers

Let us now compute the optimal number of stages in the cascade vector quantization. With (3.5) we know that

$$m_i = m/k \qquad (3.6)$$

and therefore (3.3) becomes

$$\sigma^2(k) = \sigma^2(0)\, f(m,k)^k \qquad (3.7)$$

The minimum of $\sigma^2(k)$ is achieved by

$$\frac{\partial \sigma^2(k)}{\partial k} = \sigma^2(0)\,\frac{\partial f(m,k)^k}{\partial k} = 0$$

With the identity

$$\frac{\partial f(k)^k}{\partial k} = f(k)^k\,[k\,\frac{f'(k)}{f(k)} + \log_e f(k)]$$

we know that the necessary condition is only fulfilled if the expression in brackets [.] for optimal k^* is zero:

$$[k^*\, f'(k)/\, f(k^*) + \log f(k^*)] = 0$$

or

$$k^*\, f'(k^*)/\, f(k^*) = \log 1/f(k^*) \qquad (3.8)$$

For the uniform pdf of eq.(3.1) and the identity of (3.6) this becomes

$$k^*2/nm\,(k^*/m)^{2/n\,-1}(k^*/m)^{-2/n} = 2/n = 2/n\,\log_e m/k^*$$

i.e. $\log_e m/k^* = 1,\ m/k^* = e^1 = m_i$ or $k^* = m\,e^{-1}$ (3.9)

The optimal number k^* of layers is therefore approximately 36% of the number m of available code-book vectors. Additionally, we see that the optimal number 2.7 of vectors per layer strongly supports the use of binary vector quantifiers.

Mapping each information channel to a vector quantization stage, this results fits well to the arguments of Wiener [19]. He computed for the information transfer over m channels the optimal number b of states per channel at fixed costs. With R(b,m)=bm transmission costs and $s(b,m)=b^m=b^{R/b}$ states available on the channels, the number of states and therefore the information transfer over the channels is maximized when $\partial s/\partial b=0$, i.e. $b^{R/b}[-R/b^2\ \ln b + R/b^2]=0$ is valid. This is the case for $[1-\ln b]=0$ or $b=e^1=2.71$.

Again, we conclude the optimal number of states to be approximately 2.

4 The binary cascade network

In this section we will show how the vector quantization process considered so far can be easily implemented by the means of artficial neural networks. For this purpose, let us put all the introduced elements together to form the plan of our desired network for data encoding.

With the arguments of section 2.3, we choose cascaded vector quantization for data compression with small storage needs. Section 2.1 told us that this will be facilitated by scalar vector quantization, using a PCA stage before each quantization stage. Treating all the components independently (n=1), we are told by Section 3.1 that we should choose the same number of classes per stage, which means by section 3.2 two classes for each component. Thus, we choose a binary cascaded scalar vector quantization network as main architecture for data compression and encoding.

Now, let us consider the neural network specific implementation details.

4.1 Neural networks and vector quantization

The standard vector quantization paradigm is implemented by the so-called "winner-take-all" networks. Here, several neural units receive the input in parallel and the unit with the biggest (or smallest) output trigger all the output, the others none.

For VQ, the most famous net is the Kohonen map [10] . For a large number of classes, this kind of VQ becomes problematical because it contains a non-local decision stage which needs global coordination mechanism. This is unfeasable in huge populations of neurons because they might cause timing problems by inter-neuron signal delays which is especially true in VLSI implementations.

One way to overcome this problem is the scalar vector quantization scheme. Here, we install a decorrelation stage before the quantization enabling us to treat each component independently. An efficient way to do this is a PCA layer.

4.2 Neural networks and PCA

For PCA, a lot of neural network implementations are known. For PCA, the augmented Oja subspace network [12], the GHA net of Sanger [17] or the lateral inhibition nets of Brause [2] or Rubner and Tavan [16].

All these networks decorrelate the output lines and minimize the error for neglecting less important output lines by implementing a PCA. Nevertheless, what we really need in our case is only the decorrelation property, coupled with the ability of data normalization. The latter is not obligatory, but facilitates the construction of subsequent layers and maximizes the signal to noise ratio in the processing. Here, the decorrelation nets of Silva and Almeida [18] and of Plumbley [14] are better suited. The lateral inhibited network model of Plumbley is shown in figure 4.1.

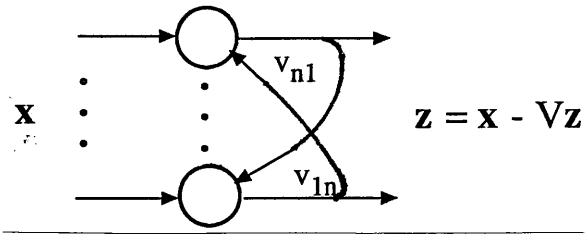

Fig. 4.1 The decorrelation and normalization network

4.3 A network for cascaded vector quantization

Let us describe now our complete network model by its activation and learning equations.

Activation architecture

For the PCA function, we choose elements of the lateral inhibited model of Plumbley as it is shown in figure 4.1. with just one scalar input for each neuron.

First, the input must be centered by an additional offset

$$x_i' = x_i - w_i \qquad (4.1)$$

Then, the input is decorrelated by the usual lateral weight influence implementing a PCA.

$$z_i = x_i' - \Sigma_j v_{ij} z_j \qquad (4.2.a)$$

or

$$z = x' - Vz \qquad (4.2.b)$$

This means for a small feedback (det(V)≤1) a convergence of the activity on the small activity time scale and a linear transform by the lateral weights

$$z\,(I+V) = x' \qquad (4.2.c)$$

or $$z = W\,x' \qquad W = (I+V)^{-1} \qquad (4.2.d)$$

The resulting signal can be easily quantized by a neuron with a binary activation function

$$S_B(z) = \begin{cases} 1 & z > 0 \\ 0 & z \le 0 \end{cases} \qquad (4.3)$$

The residual of the quantization of the 2-class intervall is obtained by producing the difference between the input x and positive and negative weights w^+ and w^-. For the positive weight w^+ this is done in the next PCA input stage by the translation $x_i \rightarrow x_i - w_i^{(k)}$ of the threshold $w_i^{(k)} = w^+$, see figure 4.2.

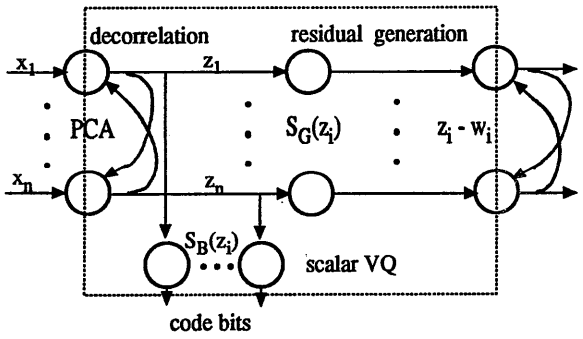

Fig.4.2 One layer of the binary cascade network

Since this works only well for positive x, we map the negative x to the positive ones by introducing the "absolute value |x|" activation function. When the input values are normalized to be less equal one, we also can use the |x|-function with the saturation limit of 1. Certainly, the negative, shifted version of this will also do the job, producing an inverted signal at each second stage.

$$S_G(z) = \begin{cases} 1-|z| & |z| < 1 \\ 0 & |z| \ge 1 \end{cases} \qquad (4.4)$$

The function $S_G(z)$ is shown in figure 4.3. Since the resulting quantization code is invertible, this coding is equivalent to the conventional one.

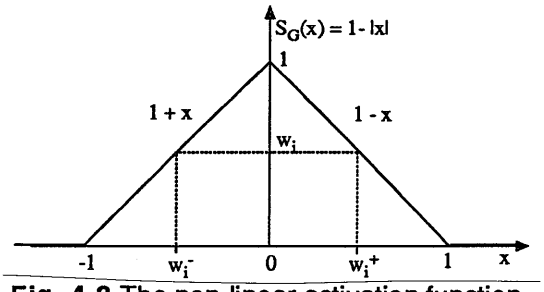

Fig. 4.3 The non-linear activation function

Please note that this non-linear activation function is a kind of bell-shaped (generalized radial basis) function defined by Cardaliaguet and Euvrard [3].

Learning rules

The first learning parameter in the network of figure 4.3 is the offset of the input x_i. To compensate it, we choose as learning goal for $w_i^{(k)}$ the expectation value $\langle x_i \rangle$. This can be achieved by several learning algorithms. For stationary sources we might choose

$$w_i^{(k)}(t+1) = w_i^{(k)}(t) + \gamma_1(t)(x_i - w_i^{(k)}) \qquad (4.5)$$

with $\gamma(t)=1/t$. This leads to equally weighted samples $x_i(t)$, see Pfaffelhuber [13]. In the non-stationary environment where the changing time constants are not known, a constant learning rate γ_1 will lead to a exponentially decreasing influence of the samples on $w_i^{(k)}$.

As learning rule of the lateral inhibition weights implementing the decorrelation, we use

$$v_{ij}(t+1) = v_{ij}(t) + \gamma_2(z_i z_j - \delta_{ij}\sigma^2) \qquad (4.6)$$

with the Kronecker symbol δ_{ij}. For i=j this corrects with $\delta_{ij}=1$ the lateral inhibition weights only when the variance z_i^2 differs from the desired variance σ^2. The motivation for this learning rule by information theoretic ideas and its PCA properties were introduced by Plumbley [14].

It should be noted that the learning (4.5) should take faster than the learning (4.6), because it is the precondition of the proper convergence of the PCA weights. This means that the learning rates γ_1 and γ_2 must be appropriately chosen, e.g. $\gamma_1 = 10\gamma_2$.

For a decoding network, the architecture should only be inversed, i.e. the residual output becomes the input to be processed. Knowing w_i and the inhibition weights v_i we can easily perform the inverse PCA by using eq. (4.2c) in simple linear neurons, including the inverse shifting.

5 Simulations

In this section we will show the characteristics of the information processing of the binary cascade network stages for the example of Gaussian distributed input pattern space processed by two layers of the network. Before, it had been trained with 10000 Gaussian distributed samples.

For each layer, the distribution in the input space $\{x^{(k)}\}$ and the centered, decorrelated (rotated) and normalized patterns $\{z^{(k)}\}$ after the PCA stage (see fig.4.2) are shown. The output patterns of the layer are identical with the input pattern of the next layer. For visualization purposes, we use only 2-dim patterns which results in 2 hidden neurons,

2 output neurons and 2 binary encoding neurons per layer.

For the whole simulation, we used γ_1=0.1 and γ_2=0.01. In figure 5.1 the space of 10000 input samples $\{x^{(1)}\}$ is shown.

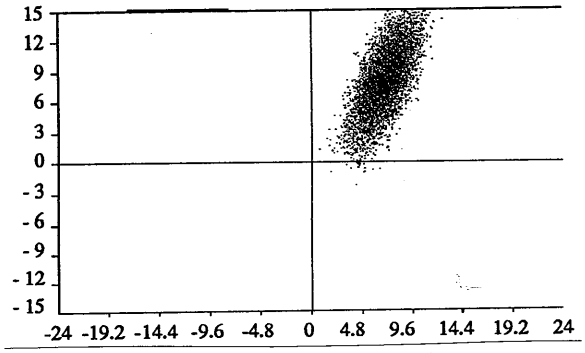

Fig. 5.1 The input pattern sample space

After being centered, rotated and normalized in the PCA stage (fig.5.2),

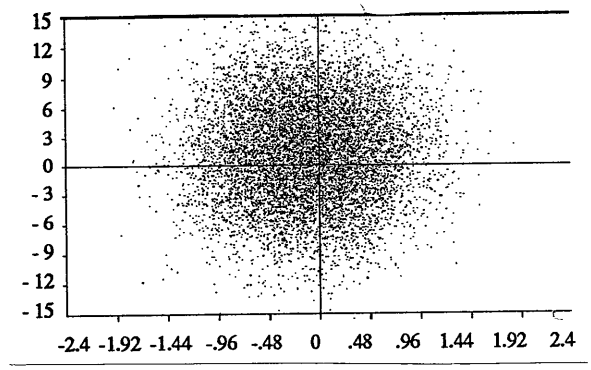

Fig. 5.2 The centered, decorrelated and normalized space

the negative parts of $\{z^{(1)}\}$ are mapped on the positive coordinates by S_G. The resulting patterns $\{x^{(2)}\}$ (fig.5.3) are fed to the next layer.

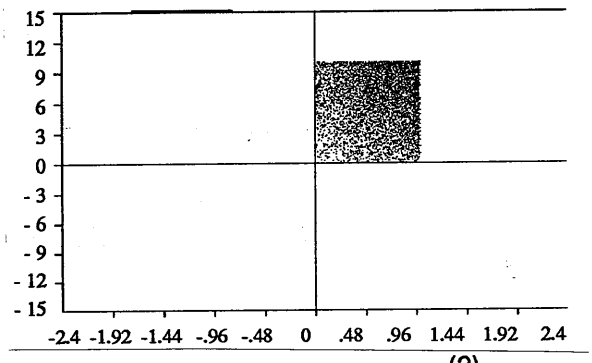

Fig. 5.3 The new distribution $\{x^{(2)}\}$

Here, the input $\{x^{(2)}\}$ is centered, rotated and normalized to $\{z^{(2)}\}$ (figure 5.4) by the second stage.

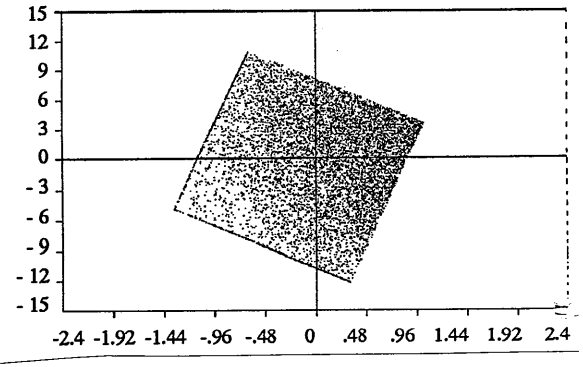

Fig. 5.4 The centered, rotated and normalized space

The pattern space $\{z^{(2)}\}$ is mapped on the residual space $\{x^{(3)}\}$ shown in figure 5.5.

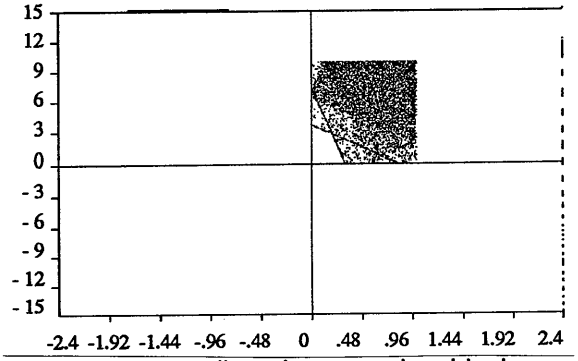

Fig. 5.5 The non-linearly mapped residual space

Gradually, we see how the initial (Gaussian) properties fade away by the subsequent non-linear mappings into a pseudo-uniform pdf.

Note: For the simulation we have computed the activity after eq.(4.2d) directly by the formula

$$W = (I+V)^{-1} = A^{-1} = \frac{1}{\det A}\,\mathrm{adj}\,(A)$$

with $\det A = A_{11}A_{22} - A_{12}A_{21}$, $A_{ij} = \delta_{ij}+v_{ij}$

and $\mathrm{adj}\,(A) = \begin{pmatrix} A_{22} & -A_{12} \\ -A_{21} & A_{11} \end{pmatrix}$

5 Discussion

Cascaded vector quantization scheme was introduced and it was shown that this allows important savings in search time and storage costs compared to the conventional approach. After the calculation of the optimal number of stages and units per stage the paper focuses on the implementation of data encoding by cascaded vector quantization using binary and non-linear PCA neural net layers without any winner-take-all mechanism.

Nevertheless, since this kind of quantization pools all residuals of one stage together to form the input space for the next stage, existing data dependecies of one stage will

be lost. The resulting quantization error increases and the performance becomes worse than the single stage quantizer according to Makhoul and al. [11].

Certainly, for the distortion measure of the mean squared error this favours the presented network only for Gaussian sources where the decorrelated components become independent. This is true for the short time statistics of pixels and speech.

But, as we regard our simulations, after some descrete quantization stages even a Gaussian source can become deformed into an pseudo uniform pdf and loose the Gaussian properties. Thus, the network performs well, implementing a deterministic vector quantization without a winner-take-all mechanisms and saving a lot of storage space and computation complexity, but these advantages should always be related to the associated gain in quantization distortion.

References

[1] S. Ahalt, A. Krishnamurthy, P. Chen, D. Melton: Competetive Learning Algorithms for Vector Quantization; Neural Networks, Vol.3, pp.277-290, 1990

[2] R. Brause: A Symmetrical Lateral Inhibition Network for PCA and Feature Decorrelation; Proc. Int. Conf. Art. Neural Netw. ICANN'93, Springer Verlag 1993, pp. 486-489

[3] P. Cardaliaguet, G. Euvrard: Approximation of a Function and its Derivative with a Neural Network; Neural Networks, Vol.5, pp.207-220 (1992)

[4] N. Farvadin, J. Modestino: Optimum quantizer performance for a class of non-Gaussian memoryless sources; IEEE Transactions on Information Theory, Vol. IT-30, No. 3, pp 485-497; May 1984

[5] A. Gersho: Asymptotically optimal block quantization; IEEE Transactions on Information Theory, Vol IT25, No. 4, pp.373-380, July 1979

[6] H. Gish, J.N. Pierce: Asymptotically efficient quantizing; IEEE Trans. Inf. Theory, Vol IT-14, No.5, pp.676-683, Sept.1968

[7] A. Habibi, P. Wintz: Image Coding by Linear Transformation and Block Quantization; IEEE Transactions on Comm. Techn., Vol. COM-19, No.1, pp.50-62, 1971

[8] J. Huang, P. Schultheiss: Block quantization of correlated Gaussian random variables; IEEE Trans. on Commun. Syst., Vol. CS-11, pp.289-296, Sept. 1963

[9] B.H. Juang, A.H. Gray: Multiple stage vector quantization for speech coding; Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, pp.597-600, 1982

[10] T. Kohonen: Analysis of a simple self-organizing process; Biol. Cyb., Vol40, pp. 135-140 (1982)

[11] J. Makhoul, S. Roucos, H. Gish: Vector Quantization in Speech coding; Proc. IEEE, Vol.73, No.11, pp. 1551-1587, 1985

[12] E. Oja, H. Ogawa, J.Wangviwattana: Principal Component Analysis by Homogeneous Neural Networks, IEICE Trans. on Inf. &Syst., Vol.E75-D, No. 3, pp. 366-382, 1992

[13] E. Pfaffelhuber, P.S. Damle: Learning and Imprinting in Stationary and Non-Stationary Environment; Kybernetik 13, pp. 229-237, 1973

[14] M. Plumbley: Efficient Information Transfer and Anti-Hebiian Neural Networks; Neural Networks, Vol.6, pp.823-833, 1993

[15] S. Roucos, R. Schwartz, J. Makhoul: Vector quantization for very-low-rate coding of speech; Proc. IEEE Global Telecomm. Conf., pp.1074-1078, 1982

[16] J. Rubner, P. Tavan: A Self-Organizing Network for Principal-Component Analysis, Europhys. Lett., 10(7), pp. 693-698 (1989).

[17] Sanger: Optimal unsupervised Learning in a Single-Layer Linear Feedforward Neural Network; Neural Networks Vol 2, pp.459-473 (1989)

[18] F. Silva, L. Almeida: A Distributed Solution for Data Orthonormalization; Proc. ICANN-91, Artificial Neural Networks, Elsevier Sc.Publ., pp.943-948 (1991)

[19] N. Wiener: Cybernetics; MIT press, Cambridge, MA, 1948

Appendix A: The expected error in a hypercube

Let us assume that we have an uniform probability distribution of events in an n-dimensional input space which is divided by a regular lattice into uniformly spaced n-dimensional hypercubes of basis length a_i in dimension i.

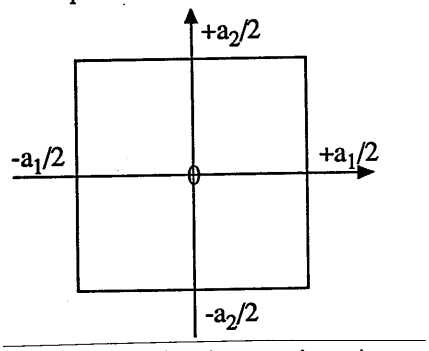

Fig. A.1 computing the error in an hypercube

Then, the expected squared error $\langle d^2 \rangle$ of all patterns in this class is

$$\langle d^2 \rangle = \int_A r^2 \, dx$$

$$= \int_{-a_1/2}^{a_1/2} \int_{-a_2/2}^{a_2/2} \dots \int_{-a_n/2}^{a_n/2} x_1^2 + x_2^2 + \dots + x_n^2 \; dx_1 \, dx_2 \dots dx_n$$

At the i-th stage, each integral gets one additive term and one constant term which, integrated for variable x_i, becomes multiplied by a.

$$\int_{-a_i/2}^{a_i/2} x_i^2 (+ \text{const}) \, dx_i = 1/3 \, x_i^3 + x_i \text{const} \Big|_{-a_i/2}^{a_i/2}$$

$$= a_i \, (a_i^2/12 + \text{const})$$

Thus, after *n* integrations, we have

$$\langle d^2 \rangle = \Pi_i a_i \, (a_1^2/12 + \dots + a_n^2/12) = \Pi_i a_i \, (\Sigma_j a_j^2/12) \qquad (A.1)$$

For equally-spaced intervals with $a_i = a_j$ this becomes

$$\langle d^2 \rangle = n \, a^n \, (a^2/12) \qquad (A.2)$$