An Adaptive Codebook Design Using the Branching Competitive Learning Network[a]

Huilin Xiong[1], Irwin King[2], and Y.S. Moon[2]

[1] Institute for Pattern Recognition and Artificial Intelligence
Huazhong University of Science and Technology, China
[2] Department of Computer Science and Engineering
The Chinese University of Hong Kong, Shatin, NT, Hong Kong
Email: {hlxiong, king, ysmoon}@cse.cuhk.edu.hk

Abstract

This paper presents an adaptive scheme for codebook design by using a self-creating neural network, called Branching Competitive Learning network. In our scheme, not only codevectors, but also codebook size are adpatively modified according to input image data and a distortion tolerance. In the situation that the input image is visually simple or the image data have a centralized distribution, our codebook design algorithm will assign a relatively small codebook; and for a complex image, our algorithm will give a relatively large codebook. Experimental results are given to illustrate the adaptability and the effectiveness of our scheme.

1 Introduction

Recently, neural network based competitive learning algorithms have been developed for vector quantization [1, 2, 3]. Compared to the conventional vector quantization algorithms, i.e., K-mean algorithm or sometimes known as Generalized Lloyd Algorithm (GLA) [4], the algorithms based on competitive learning of neural networks offer the advantages of online operation and require little storage. However, most competitive learning neural network, such as *Self-Organizing Feature Map* (SOFM) [6] and *Frequency Sensitive Competitive Learning* (FSCL) [1, 5], need to assume a network with a fixed number of nods, which means that the cluster number of input data set must be pre-specified in advance. In the situation that there is no *a priori* information available about the underlying data distribution, it is very difficult and even impossible to appropriately estimate the cluster number in a data set. As a result, we often realize only at the end of experiment that a different cluster number setting may be more proper.

In this paper, we first present a self-creating neural network model by adding a branching mechanism to the classical competitive leaning network. The proposed model, called Branching Competitive Learning (BCL) network, starts from one unit in the output layer first, and afterwards, the unit and it offspring dynamically split and merge along with the competitive learning process under control of the Branching Criteria. With this self-creating mechanism, our network can adaptively determine its size according to the input data distribution, and gives a more appropriate clustering result. As an application of the BCL network, we develop an adaptive scheme for codebook design. In our scheme, not only codevectors, but also codebook size will adaptively be modified according to the image data distribution. Experimental results are given to show the efficiency of our scheme.

2 Data Clustering By Competitive Learning

Vector quantization is a problem of data clustering. Assume there are N data vectors, $\{\vec{x}_i\}_{i=1}^{N}$ and $\vec{x}_i \in R^d$, and the codebook size (or the cluster number), M, is pre-specified, then, the codebook design can be defined as: Find $\{\vec{\omega}_i\}_{i=1}^{M}$ in R^d to minimize the average distortion or the mean squared error (MSE):

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{M}\sum_{j=1}^{N_i}\|\vec{x}_i(j) - \vec{\omega}_i\|^2, \qquad (1)$$

where N_i is data number in cluster i, $\vec{x}_i(j)$ represent data point in cluster i ($j = 1, 2, \cdots, N_i$), $N = \sum_{i=1}^{M} N_i$, and $\|\cdot\|$ is the L_2 norm.

Competitive learning network is very efficient for the task of data clustering. Kosko [8] proved that the set

[a] *This research work is supported in part by an Earmarked Research Grant (CUHK 4407/99E) from the Research Grants Council of Hong Kong, SAR.*

of centroids of clusters is the solution of the above problem, and the synaptic vectors of learning network converge to these centroids exponentially fast.

The classical Competitive Learning (CL) is a winner-take-all scheme, which can be expressed by:

$$\begin{aligned} \vec{\omega}_c(t+1) &= \vec{\omega}_c(t) + \alpha_c(\vec{x} - \vec{\omega}_c(t)) \\ &\quad \text{for } c = \arg\min_j \|\vec{x} - \vec{\omega}_j\|^2 \\ \vec{\omega}_j(t+1) &= \vec{\omega}_j(t) \qquad \text{for } j \neq c \end{aligned} \tag{2}$$

where $\vec{x}$ is a randomly selected input data point, t represents the current step of competitive learning, $\vec{\omega}_j$ denotes the synaptic vectors corresponding to the jth neural unit (for simplicity, $\vec{\omega}_j$ also represents the jth neural unit), and α_c is the learning rate. In practice, to guarantee the convergence of the learning procedure, a gradually decreasing learning rate is often adopted :

$$\alpha_c(t) = \alpha_0(1 - \frac{t}{T}), \tag{3}$$

where α_0 is the initial learning rate, and T denotes a pre-specified number of iteration.

The classical competitive learning algorithm is very simple and easily implemented; however, it suffers from the so called *dead unit* problem, which means that some units may never be activated by the competition. To deal with this problem, some modification have been developed. Kohonen's SOFM [6] employs a winner-take-quota strategy to alleviate the *dead unit* problem. In SOFM, not only the winner, but also the winner's neighbors can learn from a competition. Although SOFM can alleviate the *dead unit* problem, the performance of SOFM is greatly affected by the selection of neighborhood function, and ill-adjusted neighborhoods may lead SOFM to perform poorly.

Another modification of CL is FSCL [1, 5]. The competitive learning rule of FSCL differs from CL just by adding a winning frequency term to avoid the situation that some neurons always fail in the competition:

$$\begin{aligned} \vec{\omega}_c(t+1) &= \vec{\omega}_c(t) + \alpha_c(\vec{x} - \vec{\omega}_c(t)) \\ &\quad \text{for } c = \arg\min_j \gamma_j \|\vec{x} - \vec{\omega}_j\|^2 \end{aligned} \tag{4}$$

where γ_j is the frequency that $\vec{\omega}_j$ has won the competition up to now, that is $\gamma_j = n_j / \sum_i^t n_i$ and n_i is the cumulative number of $\vec{\omega_i}$ winning the competition. Obviously, when the winning frequency of a unit becomes large enough, the chance it continues winning will become small; contrarily, as the winning rate of a unit becomes small enough, the probability it wins the next competition will become large. In this way, FSCL achieves nearly equal node-utilization, and therefore avoids *dead units*.

3 Branching Competitive Learning Network

The above competitive learning algorithms have one thing in common: the network size M must be pre-specified in advance, or, in other words, the cluster number M for input data set need to be pre-specified. Because there is usually no *a priori* information available to appropriately estimate cluster number. We have to try to use different value of M to obtain better clustering result. An attractive way to solve this problem is to add a growing mechanism to neural network, so that network can automatically increase its size to an appropriate value according to the input data distribution. In the following, we present a self-creating scheme based on our previous work [10].

3.1 Branching Criteria and the BCL Algorithm

Intuitively, in the process of competitive learning, when a synaptic vector exhibits an intense oscillatory movement, it usually means that the vector is "attracted" by two or more different data clusters, and, at this moment, splitting this synaptic vector can notably decrease the clustering distortion MES. In the BCL scheme, we employ two geometrical measurements of winner's movement in the weight space to measure the intensity of winner's oscillation. The first one is the angle of a winner changing its moving direction in the weight space in two consecutive activations. The second is the minimal distance of winner's consecutive movements. Our Branching Criteria can be described as:

$$\begin{cases} \text{ang}(\vec{x}(t_c) - \vec{\omega}_c,\ \vec{x}(t_l) - \vec{\omega}_c) > \varphi_0 \\ \min(\|\vec{x}(t_c) - \vec{\omega}_c\|,\ \|\vec{x}(t_l) - \vec{\omega}_c\|) > d_0 \end{cases} \tag{5}$$

where $\vec{\omega_c}$ denotes a winner in current competition, $\vec{x}(t_c)$ and $\vec{x}(t_l)$ represent the two data presentation in two consecutive activation of $\vec{\omega}_c$, t_c denotes the current competition step and t_l represents the previous activated step, φ_0 and d_0 are angle and distance thresholds pre-specified to control the branching process. Geometrically, the larger of a winner changing its moving direction and longer of its movements in two consecutive activations, the more intense of the winner's oscillation. So, in the weight space as shown in Fig. 3.1, $\text{ang}(\vec{x}(t_c) - \vec{\omega}_c,\ \vec{x}(t_l) - \vec{\omega}_c)$ and $\min(\|\vec{x}(t_c) - \vec{\omega}_c\|, \|\vec{x}(t_l) - \vec{\omega}_c\|)$ can help us to measure the intensity of a winner's oscillation.

Usually, the threshold φ_0 in the Branching Criteria is set to 90°, and therefore, the Angle Criterion becomes:

$$(\vec{x}(t_c) - \vec{\omega}_c) \cdot (\vec{x}(t_l) - \vec{\omega}_c) < 0. \tag{6}$$

In practice, BCL may sporadically generate a few dead units which no longer or in a very low frequency be activated by the competitive learning. To avoid this situation, we simply introduce a threshold, β, called pruning rate, to delete the dead units: if $\gamma_j < \beta$, then

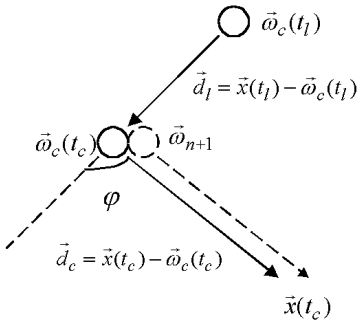
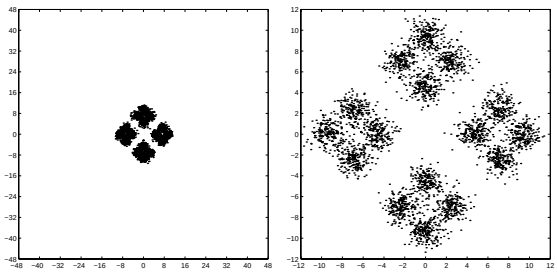


Figure 1: An illustration of branching point, where $\varphi \geq \varphi_0$ and $\min(\|\vec{d}_l\|, \|\vec{d}_c\|) \geq d_0$.



(a) (b)

Figure 2: A multiresolution data set.

delete $\vec{\omega}_j$ from the set of neural units, where γ_j denotes the frequency that $\vec{\omega}_j$ has won the competition from its birth to the current competition step. Now, let us formulate the BCL algorithm for data clustering:

1. Initialize the "seed" or the first synaptic vector.
2. Randomly take a sample $\vec{x}$ from data set, find the winner $\vec{\omega}_c$ of current competition in the set of synaptic vector $\{\vec{\omega}_j\}$ $(j = 1, 2, \cdots, n)$, that is $c = arg\ \min_j \|\vec{x} - \vec{\omega}_j\|^2$.
3. if $\vec{\omega}_c$ satisfies the Branching Criteria, a new neural unit $\vec{\omega}_{n+1}$ is spawn off from $\vec{\omega}_c$:

   $$\vec{\omega}_{n+1} = \vec{\omega}_c + \alpha_c(\vec{x} - \vec{\omega}_c)$$

   otherwise, just update $\vec{\omega}_c$ by $\vec{\omega}_c + \alpha_c(\vec{x} - \vec{\omega}_c)$.
4. For each unit $\vec{\omega}_j$, if its winning frequency γ_j from its birth time to current competition step less then the prespecified pruning rate β, i.e., $\gamma_j < \beta$, then cancel the unit $\vec{\omega}_j$ from the network.

In practice, the whole procedure can be divided into two phases: growing phase and data clustering phase. In the first phase, network keeps growing until its size reaches a dynamical equilibrium. Following this phase, network will no more split. It will just modify the synaptic vectors to approximate cluster centers.

3.2 Multiresolution Data Clustering by BCL

From a viewpoint of multiresolution, cluster number detection is resolution-dependent, and data clustering can also be viewed as problem of multiresolution spatial partition. For example, in the viewpoint of coarse resolution (Fig. 2(a)), we can say that the data set just has four clusters. However, when we view the data set in a relatively fine resolution (Fig. 2(b)), we can find four small clusters in each cluster.

In the BCL scheme, the threshold parameter d_0 plays a key role in controlling the resolution. A large value of d_0 may results in an under estimation of cluster number. From the viewpoint of multiresolution, this is natural, because a large value of d_0 means that we are viewing the data distributions with coarse resolution, in which an aggregation of locally compact clusters in the fine scale could be viewed as just one single cluster. Using different values of d_0, we can easily implement a multiresolution data clustering.

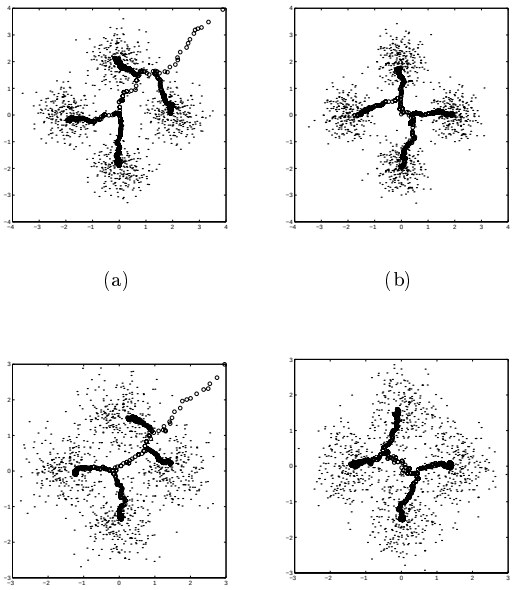
3.3 Simulations

We conduct two set of experiments on synthetic data sets in order to: (1) examine the ability of the BCL to detect cluster number automatically and to (2) show the validity of the multiresolution data clustering by the BCL.

In the first simulation, we use two sets of 2-dimensional data, as shown in Fig. 3. Each data set contains 1,000 samples with four Gaussian clusters. Figure. 3(a) and (b) show data set 1 and the learning branching traces of synaptic vectors under the "seed" initialization by point [4,4] (Fig. 3(a)), or by the origin (Fig. 3(b)). Data set 2, in which there are some overlapping between clusters, and the learning traces are shown in Fig. 3 (c) and (d).

In the experiments, we fix the learning rate with $\alpha_c = 0.05$. The threshold of angle and distance in the Branching Criteria are set with $\varphi_0 = 90°$ and $d_0 = 1.8$. The pruning rate is set with $\beta = \frac{1}{3}$. From the learning traces, we can see that BCL can automatically and correctly detect the number of clusters in data distribution, even in the case that there are some overlapping between clusters.

In the second simulation, the data set, as shown in Fig. 4, consists of 16 Gaussian distributions with standard variance $\sigma = 0.8$. Each cluster contains 200 sam-



(a) (b) (c) (d)

Figure 3: Data sets and the learning branching traces of the synaptic vectors.

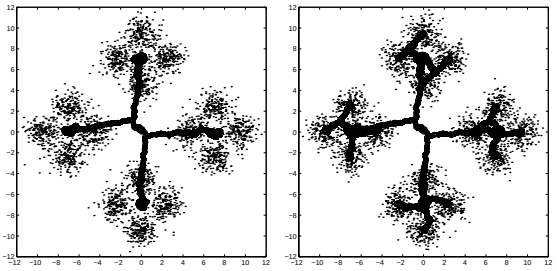
ples, and the total number of sample is 3,200.

We conduct a 2-level multiresolution clustering. In level 1, we set the distance threshold with $d_1 = 5.6$; while in level 2, the distance threshold is set with $d_2 = 2.8$. The learning rate is fixed with $\alpha = 0.02$, the original synaptic vector is initialized by point (0,0), and the pruning rate is set to $\beta = \frac{1}{3}$.

The experimental results are shown in Fig. 4. Figure 4(a) shows the learning traces of the synaptic vectors in level 1. We can see that in the resolution level 1, data points are divided into four clusters and the convergent synaptic vectors are correctly located at the centroids of each clusters. Figure 4(b) shows the learning traces of the further branching in resolution level 2. We can see that the final detected cluster number just is 16, the actual cluster number.

4 Codebook Design Using the BCL Network

In this section, we compare the performance of codebook design based on the BCL algorithm with other algorithms, such as GLA, spliting GLA [4], CL, FSCL, and RPCL [7]. The test images used here are Lena, boat, bridge, and pepper in size of 256×256 with 256 gray levels. Each image contains 4,028 image blocks



(a) Resolution level 1. (b) Resolution level 2.

Figure 4: 2-level multiresolution clustering.

with size of 4×4, and these image blocks make up a 16-dimension data set. The problem of codebook design requires that these 4,028 blocks data be coded or quantized by k blocks data, where k, the codebook size, is either pre-specified or adaptively determined according to image data distribution. As usual, we use MSE to compare the accuracy of all codebooks, MSE $= \frac{1}{256^2}\sum_{i,j}(x(i,j) - \overline{x}(i,j))^2$, where x refers to original image, and $\overline{x}$ refers to the reconstructed image.

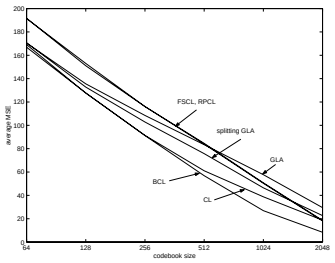
4.1 Codebook Design with Fixed Book Size

In the first experiments, the codebook size k is set to be 64, 128, 256, 512, 1024, or 2048 respectively. We adopt a piecewise random way to initialize the codebooks of GLA, CL, FSACL, and RPCL, that is, we first uniformly divide the data into k segments, then we randomly select a data sample from each segment to construct the initial codebook. As for BCL, we always initialized the "seed" by the origin point. For the competitive learning based schemes, the winner's learning rates are set in the form as in Eq. (3) with $\alpha_c(0) = 0.5$, while the rival's learning rate for RPCL is also set as in Eq. (3) with $\alpha_\gamma(0) = 0.02$. The splitting GLA algorithm is implemented as [4]. We adopt the "fractional drop of distortion" in two consecutive iterations as the convergence marker of GLA algorithm, i.e., if $\frac{D_1 - D_2}{D_1} < 0.0001$, stop the iterations of GLA, where D_1 and D_2 denote two MSEs in the previous and current iteration. For other algorithms, we just simply stop the competitive learning when the total number of learning reaches a pre-specified number. The pre-specified number is adopted in the form of mS, where m, called "epoch", is a positive integer, and S is the total number of data in the data set. In our experiments, the epoch is always set to be 20. For BCL, the pruning rate is set with $\beta = \frac{1}{6}$, and the distance threshold d_0 is set to 32.0 when $k < 2,048$, or set to 16.0 when $k = 2048$. Experimental results of average MSE and average CPU running time in ten consecutive trials are shown in Table 1 for the Lena image. Figure 5 shows the comparisons of BCL with other codebook design

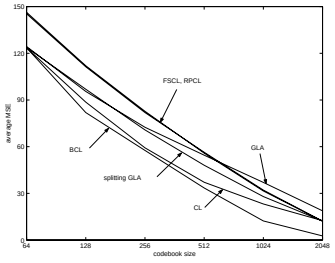
Table 1: Comparison of the algorithms for VQ design (for Lena image).

	k	64	128	256	512	1024	2048
GLA	MSE	137.16	107.75	86.85	66.39	45.98	23.25
	time(s)	42.82	89.56	155.16	201.56	354.94	491.90
split	MSE	137.16	106.97	85.68	61.06	37.68	17.06
GLA	time(s)	45.92	74.04	159.51	208.38	349.50	543.71
CL	MSE	135.20	102.43	72.58	47.50	28.73	14.22
	time(s)	20.85	42.52	84.14	167.38	337.38	737.57
FSCL	MSE	156.93	126.69	98.09	70.26	43.20	18.40
	time(s)	28.84	44.76	125.38	253.23	481.73	945.23
RPCL	MSE	155.66	125.73	97.23	69.70	43.03	17.89
	time(s)	22.41	45.78	89.61	178.18	360.33	803.39
BCL	MSE	136.35	102.53	71.97	43.65	18.42	4.88
	time(s)	16.95	33.17	65.09	132.35	376.02	817.29

algorithms in term of average MSE for the boat and pepper images.



(a) For the boat image.



(b) for the pepper image.

Figure 5: Comparison of BCL with other algorithms in term of average MSE.

From these experimental results, we can see that the BCL algorithm almost always provides a better codebook with lower MSE (especially for large codebook) than other algorithms, while, in most cases, it requires the same or even less computation time comparing with other CL algorithms.

4.2 Adaptively Determining Codebook Size

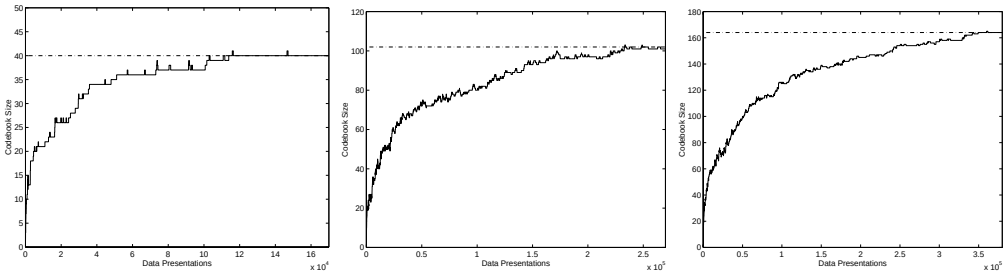
In this section, we conduct experiments to investigate the performance of BCL network for adaptively codebook design. In the experiments, the learning rate is fixed at $\alpha_c = 0.5$ in the growing phase, in which network continues growing until its size reaches a dynamical equilibrium. If the fluctuation of network size keeps in an acceptable range for a long period, the final network size is then accepted as the codebook size. After that, the network stops growing, and the synaptic vectors of the network are treated as the initial codevectors. In practice, when the fluctuation of network size S_c satisfies $\frac{|S_c(t_2)-S_c(t_1)|}{S_c(t_2)} \leq 0.01$ in three consecutive "epochs", we think the network size reaches its dynamical equilibrium. In the following competitive learning, network just modifies the codevectors to approximate cluster centers. In this procedure, learning rate is adopted in a decreasing format as Eq. 3 with $\alpha_c(0) = 0.5$.

Figure 6(a) presents three images, obviously, each of them has different visual complexity. Under a same distance threshold $d_0 = 120$, our BCL network assigns different codebooks with different size to them. Figure 6(b)$\sim$(d) show the corresponding growing phases and the final values of the codebook size. We can see that BCL can adaptively determine codebook size according to image data distribution or image's visual complexity. Besides, we also compare, in term of MSE, the adaptive codebook with other codebooks for Lena image. This time, the codebook size obtained from BCL is used as the pre-specified codebook size for other codebood design schemes. Figure 7 gives the experimental results, from which we can see that BCL network almost always generates better codebook with low MSE.

5 Conclusion

We presents an adaptive scheme for codebook design by using the Branching Competitive Learning network. In our scheme, not only codevectors, but also codebook size are adpatively modified according to input image data and a distortion tolerance. With a same distor-

(a) Original images.



(b) $S_c = 40$ (c) $S_c = 102$ (d) $S_c = 164$

Figure 6: Adaptively determined codebook.

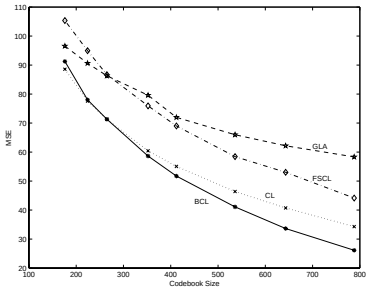


Figure 7: Comparison of adaptive BCL codebook with other codebooks.

tion tolerance, our codebook design algorithm will automatically assign a relatively small codebook to the visually simple images; while for a complex image, our algorithm will give a relatively large codebook. Experimental results are given to illustrate this adaptability and the effectiveness of our scheme.

References

[1] S. C. Ahalt, A. K. Arishnamurty, P. Chen, and D. E. Melton. Competitive learning algorithms for vector quantization. *Neural Networks*, 3:277–291, 1990.

[2] E. Yair, K. Zeger, and A. Gersho. Competitve leaning and soft competition for vector quantization design. *IEEE Trans. on Signal Processing*, 40:294–309, 1992.

[3] C. Zhu and L-M. Po. Minimax Partial Distortion Competitve Learning for Optimal Codebook Design. *IEEE trans. on Image Processing*, 7(10):1400-1409, 1998.

[4] A.Gersho and Gray R.M. *Vector Quantization and Signal Compression*. Kluwer, 1992.

[5] D. Desieno. Adding a conscience to competitive learning. *Proc. IEEE Int. Conf. Neural Networks*, I:117–124, 1988.

[6] T. Kohonen. *Self-Organization and Associative Memory*, volume 8 of *Springer Series in Information Sciences*. New York: Springer-Verlag, 1984.

[7] Lei Xu, Adam Krzyzak, and Erkki Oja. Rival penalized competetive learning for clustering analysis, rbf net, and curve detection. *IEEE Trans. on Neural Networks*, 4(4):636–649, July 1993.

[8] Bart Kosko. Stochastic competitive network. *IEEE Transactions on Neural Networks*, 15(1), 1991.

[9] N. Nasarabadi and R.A. King. Image coding using vector quantization: A review. *IEEE Trans. Commun.*, 36(8):957–971, 1988.

[10] I. King and H. Xiong. Branching Competitive Learning for Clustering. *Proceedings to the International Conference on Neural Information Processing (ICONIP2000)*, Taejon, Korea, 2000.