ADDING A CONSCIENCE TO COMPETITIVE LEARNING

Duane DeSieno
HNC, Inc.
5501 Oberlin Drive
San Diego, California 92121
(619) 546-8877

ABSTRACT

There are a number of neural networks that self-organize on the basis of what has come to be known as Kohonen learning. This paper introduces a modification of Kohonen learning that provides rapid convergence and improved representation of the input data. In many areas of pattern recognition, statistical analysis, and control, it is essential to form a non-parametric model of a probability density function p(x). The purpose of the improvement to Kohonen learning presented here is to form a better approximation of p(x). Simulation results will be presented to illustrate the operation of this competitive learning algorithm.

INTRODUCTION

Von der Malsburg (1973) developed an early model of competitive learning in his study of the visual cortex. He noted that a model of the visual cortex could not be based on genetic pre-determination but that self-organization would be required to provide the necessary plasticity demonstrated by experiments on the visual cortical cells of young kittens. Von der Malsburg's computer simulation was possibly the first to demonstrate self-organization. His model was based in part on the earlier work of Grossberg (1972) but von der Malsburg required that the weights for each processing element be normalized after each adjustment.

Grossberg (1976) developed the theory which is the basis for the Adaptive Resonance (ART) architectures. He proved that his model could provide normalization, contrast enhancement, and short term memory. In the ART1 architecture, Carpenter and Grossberg (1988) used the concept of vigilance to allow a network to establish recognition categories in a small number of passes through a training set of data. In Kohonen learning, all of the available processing elements are used, whereas in ART, the value of the vigilance parameter determines how many are used.

The neocognitron was developed by Fukushima (1982) as a neural network for performing image pattern recognition. This network uses self-organization to perform the recognition which is tolerant of deformations, scale changes, and shifts in position. In his network, only the most active processing element is reinforced by the input pattern.

The topology-preserving feature map of Kohonen (1984) developed the concept of having all the processing elements in a neighborhood N_c of the maximally responding element c reinforced. The reinforcement is accomplished by moving the weight vectors of the winning processing elements toward the input vector. As training proceeds, the size of the neighborhood decreases. Kohonen noted that a valuable characteristic of a trained network would be to have each processing element win the competition with equal probability. The examples of Kohonen learning presented here use a neighborhood of only the maximally responding element $N_c = \{c\}$.

Rumelhart and Zipser (1985) noticed that in using a competitive learning rule where only one processing element had its weights adjusted, it was possible to develop a situation in which some processing elements never win the competition. They proposed two methods to deal with this situation. The first was to allow the losing processing elements to also move toward the input vector. The learning rate for the losing processing elements would be much smaller than for the winner. The second method was to have a threshold for each processing element. If a processing element wins, it increases its threshold and when it loses, it decreases its threshold. This method forces losers toward the input data vectors and prompts the development of the conscience mechanism presented in this paper.

The most recent network using competitive learning is the counterpropagation network developed by Hecht-Nielsen (1987a). It combines Kohonen learning and Grossberg learning to produce a mapping neural network. Hecht-Nielsen noted that in the competitive layer, some processing elements could get stuck in isolated regions of the vector space. He proposed a method by which a convex combination of the data vectors with a normalized vector of ones would be presented to the network. As training progressed, the data vectors were allowed to separate, drawing the weight vectors with them. Hecht-Nielsen (1987b) also reported on my initial work in adding a conscience to the competition.

WHY A CONSCIENCE?

The most appealing characteristic of competitive learning algorithms is that they perform a vector quantization, dividing the input data vector space into discrete, equiprobable regions. Each processing element represents one of these regions in the vector space. If it is possible to have the probability that an input vector falls into a region being the same for each region, then the processing elements would generate an optimal 1 out of N code for the input vector space. It has been noted that Kohonen learning does not achieve this result, but is biased in favor of the regions of lower density of input vectors.

A further drawback of Kohonen learning is the large number of iterations required to reach a solution. Methods used to bring losing processing elements into the solution tend to increase the number of iterations required for convergence. Allowing losers to slowly adjust their weight vectors, as suggested by Rumelhart and Zipser, distorts the final regions toward the average of all the input vectors.

The goal of the conscience mechanism is to bring all the processing elements available into the solution quickly, and to bias the competition process so that each processing element can win the competition with close to the 1/N probability desired for an optimal vector quantization.

CONSCIENCE MECHANISM

The conscience mechanism consists of two parts: the generation of the outputs of the competitive layer, and a procedure for adjusting the weight vectors. When an input vector is presented to the network, a competition is held to determine which processing element's weight vector is closest in Euclidian distance to the input vector. The output y_i of the ith processing element is described as

$$y_i = 1 \text{ if } \| W_i - X \|^2 < \| W_j - X \|^2 \quad \forall_{j \neq i}$$
$$y_i = 0 \text{ otherwise.}$$

In case of a tie, the lower index wins the competition. The winning processing element in the above competition is not necessarily the element to have its weights reinforced. A bias is developed for each processing element based on the number of times it has won the above competition. Let p_i represent the fraction of time that processing element i wins the competition. An effective means of generating these numbers is:

$$p_i^{new} = p_i^{old} + B\,(y_i - p_i^{old})$$

where $0 < B << 1$.

The constant B should be picked so that p_i does not reflect the random fluctuations in the data. A B value of 0.0001 was used in the examples presented here.

Let z_i represent the winning processing element for the purposes of weight adjustment. A bias term b_i is introduced to modify the competition. Then,

$$z_i = \quad 1 \text{ if } \| W_i - X \|^2 - b_i \leq \| W_j - X \|^2 - b_j \quad \forall_j \neq i$$
$$z_i = \quad 0 \text{ otherwise,}$$

where,

$$b_i = C\,(1/N - p_i).$$

The constant C represents the bias factor and N is the number of processing elements in the competitive layer. C determines the distance a losing processing element can reach in order to enter the solution.

Finally, the weights of the processing element winning this biased competition are adjusted as follows:

$$W_i^{new} = W_i^{old} + A\,(X - W_i^{old}) z_i.$$

The constant A represents the learning rate for the weight adjustments. It represents the fraction of the distance that the winning processing element will move toward the input data vector.

As a network is trained using this algorithm, and processing elements start winning their share of the competitions, the bias terms have less impact and the process tends to revert to a simple Kohonen learning rule. By setting C to 0.0, the algorithm becomes Kohonen learning, and is therefore a generalization of Kohonen learning. The term conscience arises because a processing element that wins too often begins to "feel guilty" and prevents itself from winning excessively.

The above algorithm has a number of useful properties. Since distance is used in the competition process, normalization of the weight vectors is not necessary. The winning weight vectors can be used directly to implement a novelty filter or for use in a hierarchical network. The algorithm will work equally well on both binary and analog input data, and the weight vectors can all be initially set to 0.

SIMULATION RESULTS

The computer simulations presented in this section were designed to show the time history of the training process. Since the algorithm can work on unnormalized data, a one-dimensional problem

was chosen as an example. The author has trained networks using this method with as many as 2500 inputs to each processing element. The conscience learning law scales up linearly, making it appropriate for application to large, real-world problems.

Each training run was made on a network of 15 processing elements. The input data point X was generated randomly by taking the product of two uniformly distributed random numbers between the values of 0.0 and 1.0. The resulting probability density function is skewed heavily toward 0.0, and Figure 1 shows this density function with regions of equal probability indicated.

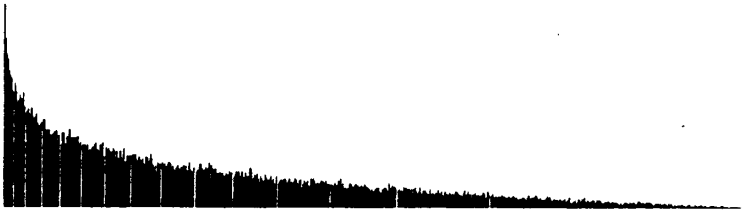

Figure 1. Probability density function showing regions of equal area.

Training runs were made for both modified Kohonen learning and conscience learning, with learning rates of A equal to 0.01, 0.03 and 0.10 for both learning laws. The conscience bias factor C was set to 10.0 for all runs where conscience was used. In all runs the weights of all 15 processing elements were initially set to 0.5. The runs are shown in Figures 2 through 7 with time moving down the page.

With a learning rate of 0.03, Figure 2 shows a Kohonen learning simulation that was allowed to train for 32,000 iterations. By the end of the run, it has still not shown the proper distribution of processing elements. However, by adding a conscience (Figure 3), conscience learning shows an equiprobable distribution of processing elements after only 3200 iterations. This represents an improvement in training time by a factor of 10.

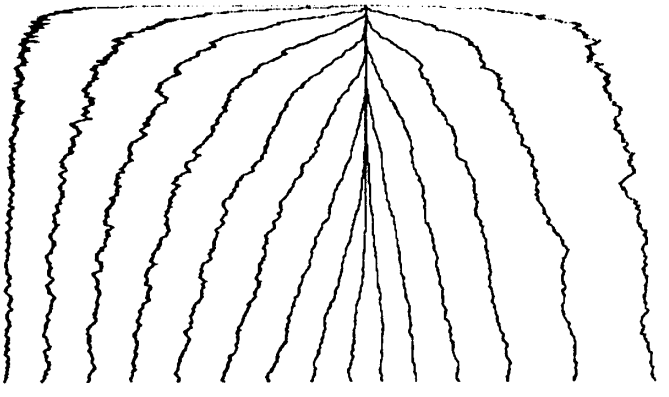

Figure 2. Kohonen learning. A = 0.03 for 32000 iterations.

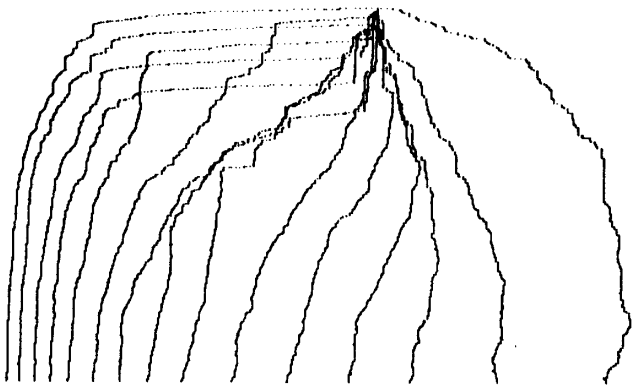

Figure 3. Conscience learning. A = 0.03 for 3200 iterations.

Note that in Figures 4 and 6 training with Kohonen learning, several processing elements had not won the competition even once by the end of the run. This is in contrast to Figures 5 and 7 in which learning with a conscience allowed each processing element to win with roughly equal probability in the same span of time.

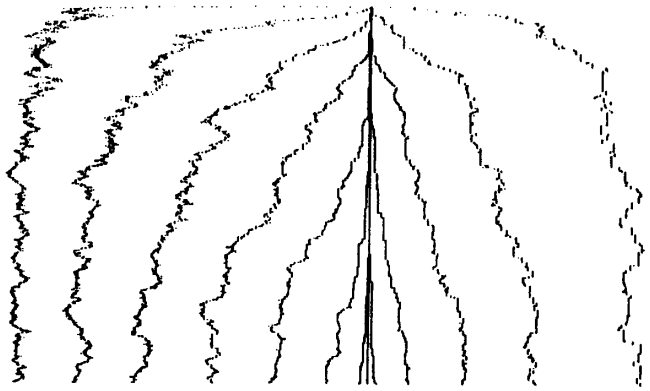

Figure 4. Kohonen learning. A = 0.10 for 3200 iterations.

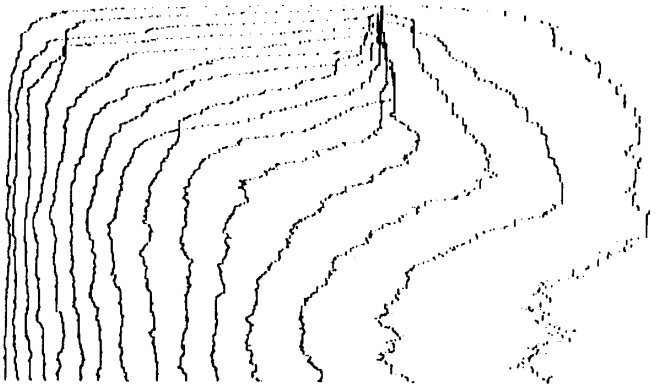

Figure 5. Conscience learning. A = 0.10 for 3200 iterations.

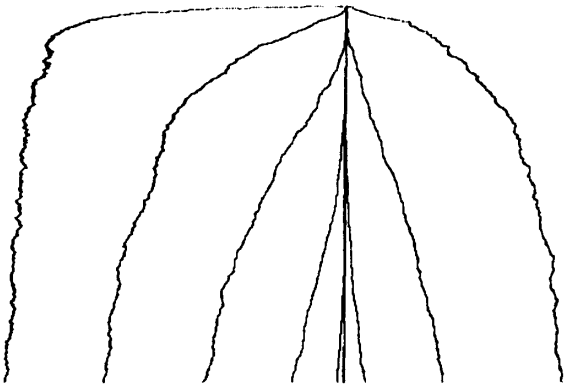

Figure 6. Kohonen learning. A = 0.01 for 9600 iterations.

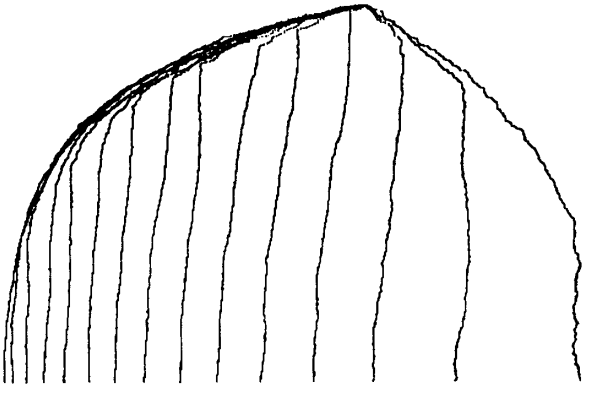

Figure 7. Conscience learning. A = 0.01 for 9600 iterations.

The final training run was made to show what happens to each algorithm when there is a gap in the probability density function. A gap was introduced between the values of 0.2 and 0.4 (see Figure 8). In practice, the input data usually clusters in a number of regions in the input vector space. Thus, the ability of a network to handle this type of problem is of great importance. As seen in Figure 9, with traditional Kohonen learning only one processing element was able to bridge the gap, and once on the left of the gap, became the only processing element capable of winning the competition. In contrast, the conscience mechanism (Figure 10) had no problem handling this type of input density function.

Figure 8. Probability density function showing regions of equal area.

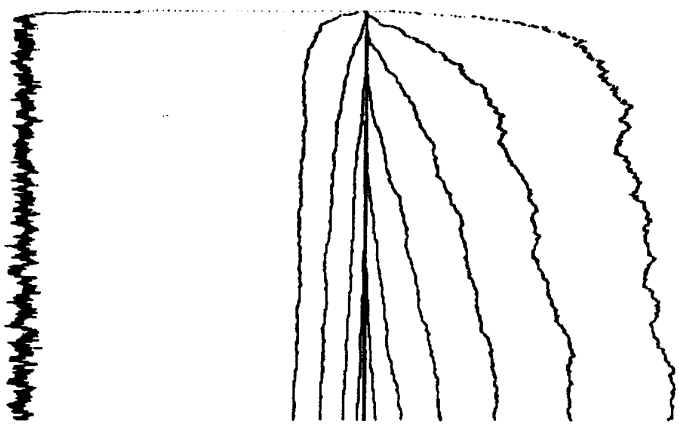

Figure 9. Kohonen learning. A = 0.03 for 16000 iterations.

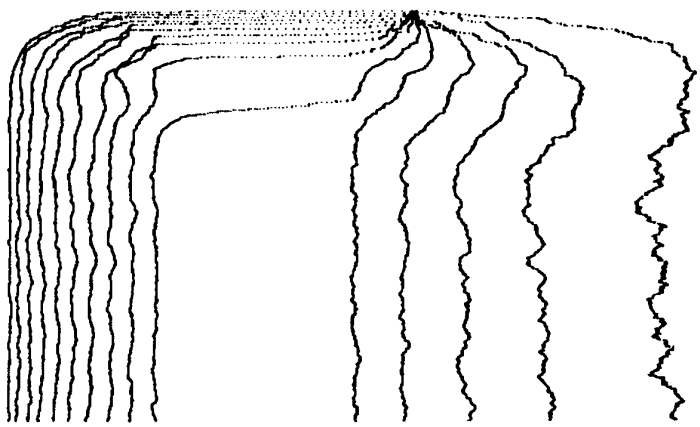

Figure 10. Conscience learning. A = 0.03 for 16000 iterations.

DISCUSSION

The conscience mechanism is effective in practice for developing a set of equiprobable features or prototypes representing the input data. While the mechanism biases the processing elements to win with equal probability even after substantial amounts of training, there is always a residual bias lowering the sensitivity of processing elements in the higher density regions of the input vector space. The bias factor C effectively controls how close to equal probability the win ratio of each processing element will come. This mechanism is a basic building block for more complex and hierarchical networks. It has been incorporated into the competitive layer for the Hecht-Nielsen counterpropagation network, and can improve the performance of most networks where competitive learning is employed.

REFERENCES

G. Carpenter and S. Grossberg, " The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network," IEEE Computer, pp. 77-88, (March 1988).

K. Fukushima and S. Miyaki, "Neocognitron: A New Algorithm for Pattern Recognition Tolerant of Deformations and Shifts in Position," Pattern Recognition Vol. 15, No. 6, pp. 455-469 (1982).

S. Grossberg, "Neural Expectation: Cerebellar and Retinal Analogs of Cells Fired by Learnable or Unlearned Pattern Classes;" Kybernetic 10, pp. 49-57, (1972).

S. Grossberg, "Adaptive Pattern Classification and Universal Recoding. I: Parallel Development and Coding of Neural Feature Detectors," Biological Cybernetics 23, pp. 121-134, (1976).

S. Grossberg, "Studies of Mind and Brain," D. Reidel, Boston (1982).

R. Hecht-Nielsen, "Counterpropagation Networks," in Proceedings, IEEE International Conference on Neural Networks, IEEE, New York (1987a).

R. Hecht-Nielsen, "Counterpropagation Networks," Applied Optics Vol. 26, No. 23 (Dec. 1987b).

T. Kohonen, "Self-Organization and Associative Memory," Springer-Verlag, New York (1984).

T. Kohonen, "The 'Neural' Phonetic Typewriter," IEEE Computer, pp. 11-22, (March 1988).

Von der Malsburg, "Self-Organization of Orientation Sensitive Cells in the Striate Cortex," Kybernetik 14, pp. 85-100 (1973).

D. Rumelhart and D. Zipser, "Feature Discovery by Competitive Learning," Cognititive Science Vol. 9, pp. 75-112 (1985).