Enhancing A Self-Organizing Map Through Near-Miss Injection

Songnian Rong and **Bir Bhanu**
College of Engineering
University of California
Riverside, California 92521

Abstract

In this paper, Kohonen's self-organizing map (SOM) is viewed from the aspect of distinguishing different classes of feature vectors. Although a well trained SOM can convert the most important similarity relationships among the input feature vectors of the same class into the spatial relationships among the responding neurons, it lacks the power to exclude the near-miss feature vectors that belong to another class. In order to use SOM as a classifier, we developed a novel algorithm called *near-miss injection* which, when used in conjunction with Kohonen's SOM algorithm, can build a more "faithful" map for a given class that covers less feature vectors from another class. A 2-class classifier is built upon the trained SOM, and experimental results are shown on synthetic data.

1 Introduction

Self-Organizing Map (SOM) is an important artificial neural network (ANN) system developed by Kohonen [1, 2]. Unlike other ANNs, whose computation powers are based only on the *weight vectors* that the neurons have learned during the training phase, the SOM utilizes both the weight vector of each neuron and the *spatial relationships* between the neurons in its computation. A well trained SOM converts the most important similarity relationships among the input data into the spatial relationships among the responding neurons. Therefore, Kohonen's self-organizing map algorithm makes it possible to discover the distribution of the training data as well as the cluster center of the data, which could be helpful for classification of feature vectors. Using the distribution information provided by a SOM, more sophisticated statistical classifiers can be constructed which would be more powerful than the conventional K-means algorithm when classifying *fuzzy* classes, i.e. classes that partially overlap in the feature space [3, 4]. In a $2D$ Kohonen's self-organizing map, neurons are arranged into an $N \times N$ array, with each neuron holding a L dimensional weight vector, where L is the dimension of the underlying feature vectors, whose distribution in the feature space is to be learned by the SOM. Given this setup, Kohonen's self-organizing process can then be described by the following algorithm ([2]):

1. *Initialization*: Start with appropriate initial values for the weight vectors $\mathbf{w}_{rl}$, where r and l are the row and column indexes of the neuron in the SOM, respectively.
2. *Choice of Stimulus*: Randomly choose, according to the probability density of the training vectors, an input vector $\mathbf{v}$.
3. *Response*: Determine the corresponding "hitting neuron" $\mathbf{c}$ from the condition
$$||\mathbf{v} - \mathbf{w_c}|| = \min_{\mathbf{i}} ||\mathbf{v} - \mathbf{w_i}||, \text{ where } \mathbf{i} = (\mathrm{r,l}),\ \mathrm{r} = 1, 2, \cdots, N;\ \mathrm{l} = 1, 2, \cdots, N \tag{1}$$
4. *Adaptation*: Update the weight vectors of the hitting neuron and every neuron within a neighborhood of the hitting neuron according to
$$\mathbf{w_i}(t+1) = \begin{cases} \mathbf{w_i}(t) + \alpha(t)\left(\mathbf{v}(t) - \mathbf{w_i}(t)\right) & \text{for } \mathbf{i} = (\mathrm{r,l}) \in N_c \\ \mathbf{w_i}(t) & \text{otherwise} \end{cases} \tag{2}$$

where N_c denotes the neighborhood of $\mathbf{c}$, and $\alpha(t)$ is the learning rate.

5. *Termination Checking*: If the SOM is well organized, terminate the training process. Otherwise, go to step 1.

Different strategies can be used to control the learning rate $\alpha(t)$ and to adjust the neighborhood N_c as training proceeds. Both α and N_c should decay with time. In the above algorithm, the training process normally terminates when a pre-selected iteration number has been reached. The selection of this number is mainly based on experiments.

2 Learning From Positive And Near-miss Examples

In order to solve the feature overlapping problem that accompanies most classification problems, a novel algorithm called *near-miss injection* is developed to enhance the SOM algorithm. The first step of our algorithm is to use the Kohonen's algorithm to train the SOM by using positive training examples. By positive examples we mean feature vectors from a class that has been selected as positive class.

2.1 Disorder Index

When applying Kohonen's algorithm to real-world problems, it requires a lot of experiments to select a good set of parameters. The termination criterion is one of them. To make the learning process autonomous, i.e. without the need for humans intervention, a metric reflecting the SOM's ordering is needed so that the algorithm can determine how well the SOM has been trained, and thus determines whether it is time to terminate the learning process. In our research, we developed two metrics to describe the ordering of a SOM. The first one is based on the proved asymptotic convergence property of the SOM, and the second one is based on a direct analysis of the geometric distortion of the SOM.

Disorder index 1

Since a properly trained SOM asymptotically converges to the distribution of training examples, the variation of the weight vectors with respect to a fixed number of training iterations will decrease asymptotically. So a measure based on this variation can be used as an index for the ordering of SOM. Let $d_{ms}(t)$ be the mean square distance between the training vectors and the weight vectors at discrete training time t, we have

$$d_{ms}(t) = \frac{1}{|S_T|} \sum_{\mathbf{x} \in S_T} \left(\frac{1}{|N_c|} \sum_{i \in N_c} ||\mathbf{x} - \mathbf{w}_i(t)||^2 \right) \tag{3}$$

where S_T denotes the training example set and N_c is the set of neurons within the neighborhood of the hitting neuron. The *Disorder Index* (DOI) can then be defined as

$$DOI = d_{ms}(t+k) - d_{ms}(t) \tag{4}$$

where k determines the length of the interval when DOI is evaluated.

Disorder index 2

Another approach to evaluate the disorder degree of a SOM is to directly analyze the distortion of the SOM. If we keep the SOM to be of the same dimension as the feature vectors (this is not required by the SOM algorithm, it is solely for the application of the following disorder analysis), the disorder index can be obtained from analyzing the distortion of $2D$ *neuron grid* or the $3D$ *neuron cube*, depending on the dimension of the feature vectors. For the $2D$ case, we say a SOM is well ordered if it satisfies the following conditions:

- All the extreme neurons must be boundary neurons.
- The ratio between the number of distorted grids and the number of total grids in the SOM must be less than a preselected threshold.

Although we need a preselected threshold to apply this disorder index, it is directly related to the ordering of the SOM, thus is easier to select compared with the total number of iterations. This disorder index can be formulated as following:

$$DOI = \begin{cases} Th + 1 & \text{if } \{boundary\} \neq \{extreme\} \\ Nd/Nt & \text{if } \{boundary\} = \{extreme\} \end{cases} \tag{5}$$

where Th is the preselected threshold for this disorder index, Nd is the number of distorted grids, and Nt is the total number of grids in the SOM. $\{boundary\}$ is the set of boundary neurons and $\{extreme\}$ is the

set of the extreme neurons. A neuron is identified as an extreme neuron if at least one element of its weight vector holds an extreme value. Finding out distorted neuron grids is done by analyzing the geometric relation between the four edges of a grid. For $3D$ cases, the relation between the six faces of a neuron cube must be analyzed before we can say if the neuron cube is distorted or not. For cases of even higher dimensions, the relation would be between the hyperfaces of a hypercube.

2.2 Near-miss Injection

When the disorder index is below a pre-selected threshold the SOM is in a well ordered state, and a conventional SOM algorithm can terminate its learning process. In our algorithm, at this time the learning process will go into the second stage — refining those ambiguous regions in the SOM by using the *near-miss injection* algorithm and negative examples. By ambiguous regions we mean regions where features of different classes overlap. The near-miss injection algorithm runs inside a loop which contains the following two steps:

1. given a near-miss vector $\mathbf{y}$, search the "hitting" neuron h using equation (1).
2. update the weight vectors according to

$$\mathbf{w_i}(t+1) = \begin{cases} \mathbf{w_i}(t) + \frac{\beta(t)\mathbf{u}}{(||\mathbf{y}-\mathbf{w_i}(t)||+1)^2} & \text{for } \mathbf{i} \in N_h \\ \mathbf{w_i}(t) & \text{otherwise} \end{cases} \tag{6}$$

$$\mathbf{u} = \begin{cases} \frac{\mathbf{y}(t)-\mathbf{w_i}(t)}{||\mathbf{y}(t)-\mathbf{w_i}(t)||} & \text{if } ||\mathbf{y}-\mathbf{w_i}(t)|| \neq 0 \\ \frac{\mathbf{y}(t)-\bar{\mathbf{w}}_\mathbf{i}(t)}{||\mathbf{y}(t)-\mathbf{w_i}(t)||} & \text{if } ||\mathbf{y}-\mathbf{w_i}(t)|| = 0 \end{cases} \tag{7}$$

$$\bar{\mathbf{w}}_\mathbf{i} = \frac{1}{4}\sum_j \mathbf{w_j}, \text{ neuron } \mathbf{j} \in \text{4-neighbor of neuron } \mathbf{i} \tag{8}$$

where N_h is the predefined neighborhood of the hitting neuron. $\beta(t)$ is the learning rate for this near-miss injection algorithm, it should decay with time, and we use the same decay function that is used in Kohonen's algorithm to control β.

2.3 Self-Organizing Map As A Classifier

For a 2-class classification problem (i.e. positive vs. negative), one class is selected as the positive class, and a SOM is trained (1) by applying the Kohonen's algorithm with the training examples from the positive class, and (2) by applying our near-miss injection algorithm with examples that come from the negative class. A classification criterion is then constructed based on the computation of two coefficients, $Conf_P$ and $Conf_N$, which are the confidence values for a testing feature vector to belong to the positive and negative class, respectively. These two confidence values are computed by comparing the *Four Neighbor Average Distance (FNAD)* of a testing feature vector to $\bar{R}_P$, the average *FNAD* of all the positive training examples, and $\bar{R}_N$, the average *FNAD* of all the near-miss examples. To compute the *FNAD* of a feature vector, the feature vector is first projected into the learned SOM. The hitting neuron is found next, and the four neighbors of this hitting neuron are identified. The *FNAD* can then be calculated according to

$$FNAD = \frac{\sum_{i=1}^{4} r_i}{4} \tag{9}$$

where r_i is the Euclidean distance between the testing feature vector and one of the 4-Neighbor neurons. Figure 1 shows the projection of the testing feature vector into the SOM. From Equation (9) we have

$$\bar{R}_P = \frac{\sum_{n=1}^{N_P}\sum_{i=1}^{4} r_i^n}{4N_P} \tag{10}$$

$$\bar{R}_N = \frac{\sum_{n=1}^{N_N}\sum_{i=1}^{4} r_i^n}{4N_N} \tag{11}$$

where N_P is the number of positive training examples used in the learning, and N_N is the number of near-miss examples. The two confidence values are

$$Conf_P = \max\left(1, \frac{4\bar{R}_P}{\sum_{i=1}^{4} r_i}\right) \tag{12}$$

$$Conf_N = \max\left(1, \frac{\sum_{i=1}^{4} r_i}{4\bar{R}_N}\right) \tag{13}$$

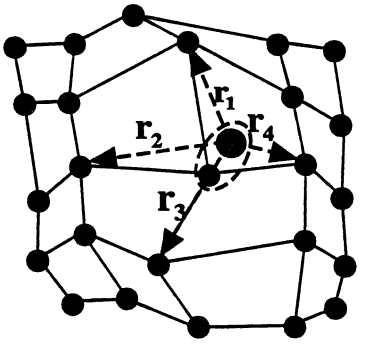

Figure 1: Classifying a testing feature vector (shown as the shaded circle) by projecting it into the trained SOM. The oval of broken line illustrates the selection of the hitting neuron.

Given $Conf_P$ and $Conf_N$, the classification of the testing feature vector is obtained as,

$$l = \begin{cases} Positive & \text{if } Conf_P > Conf_N \\ Negative & \text{Otherwise} \end{cases} \tag{14}$$

The classification confidence C_C is assigned the value of $Conf_P$ or $Conf_N$ accordingly.

3 Experimental Results

In our experiment, we compared Kohonen's algorithm with our algorithm that uses both positive and negative examples in the training phase. Figure 2 shows the synthetic data used in our experiment. Each data point in the Figure represents a synthetic 2-dimensional feature vector. Positive examples are evenly distributed over a 2×2 square area, with a hole (of radius 0.5) in the center. Negative examples are evenly distributed in a circular area with a radius of 0.7. The overlapping region is a ring of width 0.2. The size of the SOM used in the experiment is 7×7. Figure 3(a)(b) shows the trained SOM after applying Kohonen's algorithm for 1000 epochs. By applying the classification criterion given in Section 2.3, each synthetic feature vector is classified by being projected into the 7×7 self-organizing map. Out of 795 positive examples, 171 are misclassified as negative, while out of 192 negative examples, 45 are misclassified as positive. Figure 3(c)(d) shows the result of applying our algorithm to the same data set for 1000 epochs. The misclassification errors drop from 171 to 114 and 45 to 43, respectively. This result shows that the near-miss injection helps to capture a more faithful distribution of the testing data than Kohonen's algorithm, given that the feature vectors of the two classes partially overlap in the feature space.

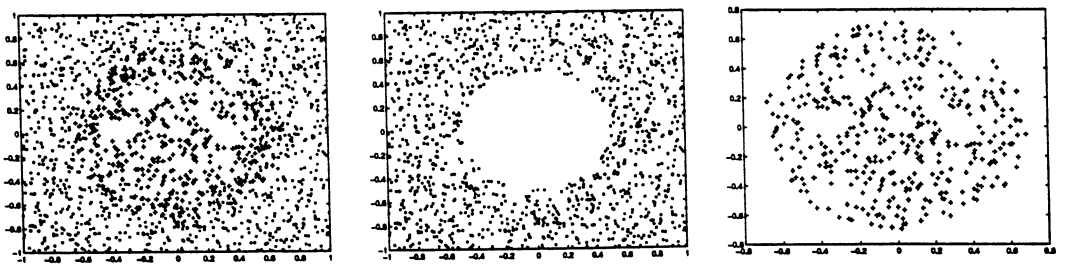

Figure 2: Distribution of the synthetic data used to compare our supervised SOM algorithm with Kohonen's algorithm. Positive and negative examples are denoted by dot (.) and plus (+) signs, respectively. (a) Positive examples overlapped with negative examples. (b) Positive examples. (c) Negative examples. The figures are not drawn to scale.

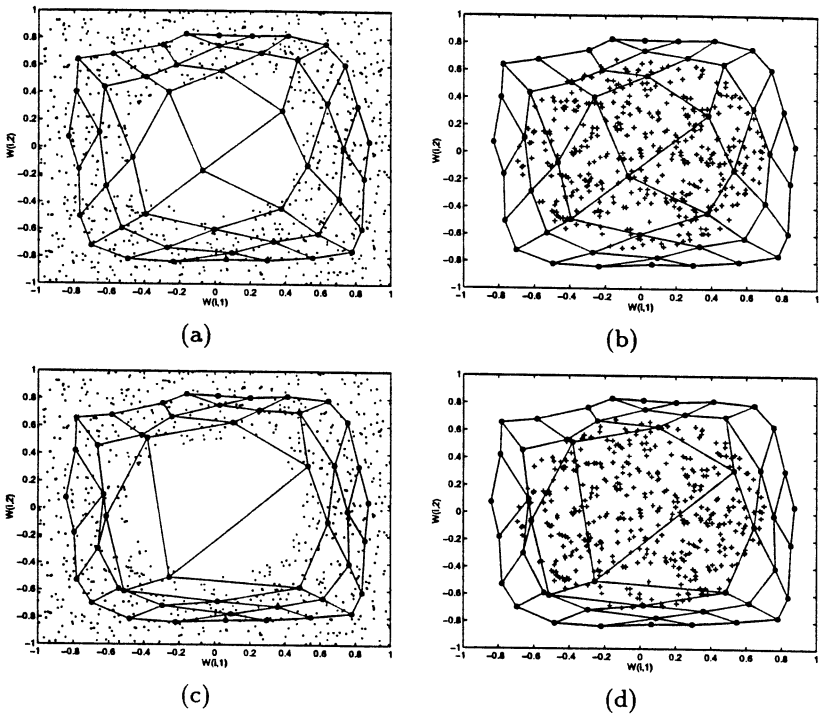

Figure 3: (a) and (b): SOM after applying Kohonen's algorithm for 1000 epochs. (a) SOM overlapped with positive examples which are shown as dots (.). (b) SOM overlapped with negative examples which are plotted as plus signs (+). (c), (d): SOM after applying our algorithm for 1000 epochs on the same training data set. (c) SOM overlapped with positive examples which are shown as dots (.). (d) SOM overlapped with negative examples which are plotted as plus signs (+).

4 Conclusions

Our work on enhancing Kohonen's SOM algorithm through near-miss injection has shown that self-organizing maps can be used as a effective classifier to distinguish different classes of feature vectors. The learning ability of the self-organizing map and the "trimming" power of the near-miss injection algorithm make SOMs more suitable for classification of real-world data, which often times are very hard to be described by any theoretical distribution functions. In this article we only discussed the 2-class classification problem. Multi-class classification problems are still being investigated.

References

[1] T. Kohonen. Adaptive, associative, and self-organizing functions in neural computing. *Applied Optics*, 26(23):4910–4918, 1987.

[2] T. Kohonen. *Self-Organizing and Associative Memory.* Springer-Verlag, Berlin, 1989.

[3] S. Mitra and S. K. Pal. Self-organizing neural network as a fuzzy classifier. *IEEE Trans. Systems, Man, and Cybernetics*, 24(3):385–399, 1994.

[4] E. Maillard, B. Zerr and J. Merckle. Classification of texture by an association between a perceptron and a self-organizing feature map. In *Proc. Sixth European Signal Processing Conference*, pages 1173–1176, August 1992.

WCNN'95 – WASHINGTON, D.C.

WORLD CONGRESS ON NEURAL NETWORKS

1995 International
Neural Network Society
Annual Meeting

Renaissance Hotel
Washington, D.C., USA
July 17 - 21, 1995

Volume I

Sponsored by the International Neural Network Society