A Supervised Self-Organizing Map for Structures

Markus Hagenbuchner
Information Technology Services
University of Wollongong, Australia
markus@artificial-neural.net

Ah Chung Tsoi
Office of Pro Vice-Chancellor (IT)
University of Wollongong, Australia
act@artificial-neural.net

***Abstract*— This paper proposes an improvement of a supervised learning technique for Self Organizing Maps. The ideas presented in this paper differ from Kohonen's approach to supervision in that a.) a rejection term is used, and b.) rejection affects the training only locally. This approach produces superior results because it does not affect network weights globally, and hence, prevents the addition of noise to the learning process of remote neurons. We implemented the ideas into Self-Organizing Maps for Structured Data (SOM-SD) which is a more general form of Self-Organizing Maps capable of processing graphs. The capabilities of the proposed ideas are demonstrated by utilizing a relatively large real world learning problem from the area of image recognition. It will be shown that the proposed method produces better classification performances while being more robust and flexible than other supervised approaches to SOM.**

I. Introduction

Self Organizing Maps (SOMs), introduced by Kohonen [1], are a well known neural model and are popular in areas that require visualization and dimension reduction of large, high dimensional data sets. SOMs are a vector quantization method which can preserve the topological relationships between input vectors when projected to a lower dimensional display space. The SOM was developed to help identify clusters in multidimensional datasets. The SOM does this by effectively packing the dataset onto a q-dimensional plane where often $q = 2$. The result is that data points that are "similar" to each other in the original multidimensional data space are then mapped onto nearby areas of the q-dimensional output space. SOMs combine competitive learning with dimensionality reduction by smoothing the clusters with respect to an a priori grid. The SOM is called a topology-preserving map because there is a topological structure imposed on the nodes in the network. A topological map is simply a mapping that preserves neighbourhood relations.

The basic idea of SOM is simple. Every neuron i of the map is associated with an n-dimensional codebook vector $\mathbf{m}_i = (m_{i1}, \ldots, m_{in})^T$. The neurons of the map are connected to adjacent neurons by a neighbourhood relation, which defines the topology, or the structure, of the map. Common topologies are rectangular and hexagonal [1].

Adjacent neurons belong to the neighbourhood N_i of the neuron i. Neurons belonging to N_i are updated according to a neighbourhood function $f(.)$. Most often, $f(.)$ is a *Gaussian-bell* function. Typically, the topology and the number of neurons remain fixed from the beginning though this is not a limitation [2]. The number of neurons determines the granularity of the mapping, which has an effect on the accuracy and generalization of the SOM [1].

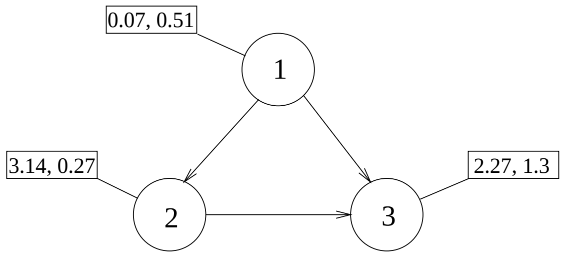
The network is trained by finding the codebook vector which is most similar to an input vector. This codebook vector, and its neighbours are then updated so as to render them more similar to the input vector. The result is that, during the training phase, the SOM forms an elastic cover that is shaped by input data. The algorithm controls the cover so that it strives to approximate the density of the data. The reference vectors in the codebook drift to areas where the density of the input data is high. Eventually, only few codebook vectors lie in areas where the input data is sparse [1].

The standard SOM model is restricted to the processing of fixed sized input vectors though extensions to data sequences do exist [3]. A more recent extension, called SOM for Structured Data (SOM-SD) [4], allows the discovery of similarities among more complex objects such as labelled acyclic directed graphs (Labelled DAGs). The extension is made possible by including structural information about the graph to the input of the network.

These methods are typically trained in an unsupervised fashion even if a teacher signal is available. Some attempts were made to utilize teacher signals, if available, with the goal to improve the mapping precision. Supervised training of SOMs is attempted by Kohonen in [1].There, supervision is achieved by attaching information about class membership to the input vector while during the recognition phase, the class label is omitted. The idea of attaching a class label to the input vectors has also been adopted to the training of SOM-SDs in [5]. While it is claimed that the attachment of class information leads to a better discrimination between pattern classes, we found in practice that this is not always true. In particular, the mapping of large sets of concise data may actually worsen. We attribute this mainly to the following two factors:

A.) For every input vector, the SOM is updated globally. Hence, the processing of an input belonging to a class 'a' adds noise to the mapping of another input from another class. The effect worsens the greater the number of different classes, and if the number of training samples for each class is out of balance.

B.) The influence of the attached class label to the error measure is unbalanced. For example, when training a SOM on inputs with a small data label and a large class label, the mapping of the input is biased towards the class label resulting in a poor representation of the actual data.



Node id	Data label	Children
1	(0.07, 0.51)	2,3
2	(3.14, 0.27)	3
3	(2.27, 1.30)	

Fig. 1. An example of a labelled directed graph (top), and the same graph in tabular representation (below).

In [5], the problem described under item **B** is overcome by weighting the influence of the data label and the class label during network training. However, this still produces good results only when the number of different classes is small.

An improvement is achieved in [6] where a rejection term is introduced to the learning process. This is made possible by dynamically assigning neurons in the map to a class. Rejection is performed when the best matching neuron and the input data belong to different classes. The approach also allows a more sophisticated control over the amount of supervision to the learning process. However, network weights are affected globally during rejection which introduces noise to the learning process of remote neurons.

In this paper we suggest to reject codebooks only locally on a cluster of neurons which are located near the best matching neuron and which belong to the same class as the best matching neuron. This eliminates both problems, **A** and **B**, allowing an application of SOM to a wide range of application domain. We will demonstrate the capabilities through the application to a large set of graph-structured real world data.

The structure of this paper is as follows: In Section II the SOM-SD model is described in some detail. This forms the basis for Section III where we present the supervised approach. Experimental results are presented in Section IV. Finally, conclusions are drawn in Section V.

II. A SOM for Data Structures

In [4], a SOM capable of processing labelled directed acyclic graphs (DAGs) is described. This extended SOM is called SOM for Structured Data, or SOM-SD. As is demonstrated in [4], a SOM-SD model defines a general mechanism which includes standard SOMs as a special case.

A SOM becomes a SOM-SD through an extension in the training algorithm, and through an appropriate presentation of DAGs to the network: The following example helps to illustrate the mechanism: Figure 1 shows a graph with 3 nodes. Each node in the graph is identified by a unique symbol (here numbers). Associated with each node is a 2-dimensional real valued data label. The node numbered 1 is called *root*, node 3 is the *leaf* node of this graph. All other nodes are *intermediate* nodes. The maximum out-degree (the maximum number of children at any node) of this graph is 2. A vectorial representation is obtain through a transformation into a tabular form as demonstrated in Figure 1. These vectors can be made constant in size if the maximum out-degree and the maximum data label dimension are known. For nodes with missing children or smaller data label size, padding with a suitable constant value can be deployed.

Traditionally, SOMs use fixed size data labels as input but structural relationships between the data is not considered. In the case of graphs, relationships between the data vectors is well defined, and hence, can be used to assist network training.

In SOM-SD this is achieved by adding the spatial location, i.e., the location of the winning neuron of the offsprings to the set of features of the parent node. As a result, the network needs to know where the winning neurons for all children nodes are located on the map when processing the parent node. This forces the network to process data in a bottom-up fashion, from the leaf nodes to the root node.

In practice, for each node in the graph, the network input will be vectors which consist of **(a)** the p-dimensional data label $\mathbf{l}$, and **(b)** the coordinates $\mathbf{c}$ of the winning neuron for each offspring.

The vector $\mathbf{c}$ is qo-dimensional, where o the maximum out-degree of any graph in the data set, and q is the dimension of the map. In this paper we will use $q = 2$. Hence, $\mathbf{c}$ consists of o tuples, each tuple being a 2-dimensional vector representing the x-y coordinates of the winning neuron of an offspring node. Offsprings, which essentially are sub-graphs, are represented at a particular position within the map. Hence, the tuples in $\mathbf{c}$ contain the coordinates of codebook vectors which were associated with the offsprings of the current node. Once it is known where the children are represented on the map, we can update the vector component $\mathbf{c}$ of the parent node accordingly. The input vector $\mathbf{x}$ is built through the concatenation of $\mathbf{l}$ and $\mathbf{c}$ so that $\mathbf{x} = [\mathbf{l}, \mathbf{c}]$. As a result, $\mathbf{x}$ is a $n = p+2o$ dimensional vector. The codebook vectors $\mathbf{m}$ are of the same dimension.

The network architecture of a SOM-SD network corresponds to that of a classic SOM but the training algorithm differs. The difference of the training algorithm arises out of the fact that the network input $\mathbf{x}$ consists of two components $\mathbf{l}$ and $\mathbf{c}$. It is required to modify the similarity measure (e.g. the Euclidean distance) so as to weight the influence of $\mathbf{l}$ and $\mathbf{c}$. This weighting is necessary to balance the influence of the elements to the training algorithm. For example, $\mathbf{l}$ may be high dimensional with elements larger than in $\mathbf{c}$. Without balancing components, the network would be unable to learn relevant information provided through $\mathbf{c}$.

III. Supervised SOM for structured information

Kohonen describes in [1] a mechanism for training a SOM supervised. The idea is to produce input vectors through the concatenation of the (numeric) target vector with the data label, and then to proceed training in the usual manner. However, the approach produces good results only for some artificial learning tasks where the number of classes is small [5]. This section describes a method for supervised training of SOM which has been developed with considerably more success.

An extension of the SOM-SD algorithm [6], employs a teacher signal when processing nodes for which a (symbolic) target label exists. The idea is to assign codebook vectors to the same class as the node that was mapped at this location. Training proceeds in a similar manner as the unsupervised case with the difference that codebook entries are *rejected* if they belong to a different class.

In detail the method works follows: Given a self organizing map $\mathbf{M}$ with k neurons. Each neuron is associated with a codebook entry $\mathbf{m} \in I\!R^n$. The best matching neuron $\mathbf{m}_r$ for an input $\mathbf{x}$ is obtained e.g., by using the Euclidean distance:

$$r = \arg\min_i \|(\mathbf{x} - \mathbf{m}_i)\mathbf{\Lambda}\| \tag{1}$$

where $\mathbf{\Lambda}$ is a $n \times n$ dimensional diagonal matrix. Its diagonal elements $\lambda_{11} \cdots \lambda_{pp}$ are assigned to be μ_1, all remaining diagonal elements are set to μ_2. The values μ_1 and μ_2 weight the influence of the components $\mathbf{l}$ and $\mathbf{c}$ in $\mathbf{x}$. Then the i-th element of the j-th codebook vector $\mathbf{m}_j$ is updated as follows:

$$\Delta m_{ij} = \begin{cases} -\beta\ \alpha(t)\ f(\Delta_{jr}) h(x_i, m_{ij}) & \text{if } \mathbf{x} \text{ and } \mathbf{m}_r \text{ are in different classes.} \\ \alpha(t)\ f(\Delta_{jr})(x_i - m_{ij}) & \text{else.} \end{cases} \tag{2}$$

$f(\Delta_{jr})$ is a neighbourhood function which will be explained later, α is the learning rate which decreases linearly to zero in time, β is a rejection rate which weights the influence of the rejection term $h(.)$. The purpose of the rejection term is to move $\mathbf{m}_r$ and its neighbours away from $\mathbf{x}$. The effect is a reduction of the likelihood that an input node activates a codebook vector which is assigned to a foreign class in subsequent iterations. The rejection term is defined as follows:

$$h(x_i, m_{ij}) = \text{sgn}(x_i - m_{ij})(\rho_i - |x_i - m_{ij}|) \tag{3}$$

where sgn$(.)$ is the signum function returning the sign of its argument, and ρ_i is the standard deviation defined as follows:

$$\rho_i = \sqrt{\frac{\sum_{l=1}^{N}(x_{li} - \overline{x_i})^2}{N}} \tag{4}$$

where N is the number of nodes in the training set, and $\overline{x_i} = 1/N \sum_{l=1}^{N} x_{li}$. ρ_i can be approximated by a constant when assuming that the mapping of nodes is random. This approximation significantly reduces the computational cost.
The rejection term dictates stronger actions if a codebook entry is very similar to the input node. Note that $h(.)$ returns values within the range $[-1; 1]$ eliminating the harmful influence of the magnitude of the vector elements.

The neighbourhood function $f(.)$ controls the amount by which the weights of the neighbouring neurons are updated. The neighbourhood function $f(.)$ can take the form of a Gaussian function:

$$f(\Delta_{ir}) \quad = \quad \exp\left(-\frac{\|\mathbf{l}_i - \mathbf{l}_r\|^2}{2\sigma(t)^2}\right) \tag{5}$$

where $\sigma(t)$ is the spread decreasing with the number of iterations[1], and $\mathbf{l}_r$ is the location of the winning neuron, and $\mathbf{l}_i$ is the location of the i-th neuron in the lattice. Other neighbourhood functions are also possible [1].

Note that because of Equation 2, the weights will not be distributed as a linear function of the input density. The approach with Equation 3 resembles MAE quantization.

Training a supervised SOM-SD network is an extension of the training algorithm used by an unsupervised SOM-SD network. The difference between the two approaches is that codebook vectors are assigned to a class during training, and the use of a rejection term for codebook entries that do not belong to the same class as the input vector. Training a SOM-SD network in a supervised fashion is as follows:

Step 1: A node j from the training set is chosen and presented to the network. When choosing a node special care has to be taken that the children of that node have already been processed. Hence, at the beginning of this process terminal nodes for each graph are processed first, the root node is considered last. Then, vector $\mathbf{x}_j$ is presented to the network. A winning neuron r is obtained by finding the most similar codebook entry $\mathbf{m}_r$, e.g. by using Eq. 1. Then, the winning neuron is assigned to the same class as the node. Step 1 is repeated until all nodes in the training set have been considered exactly once. Codebook vectors that were activated by nodes belonging to different classes are assigned to the class which activated this neuron most frequently. Neurons that were not activated by any node are assigned to the class *unknown*. Note, this step does not involve any training. It is solely used to initialize neurons with class labels.

Step 2: A node j is chosen from the data set in the same way as Step 1. Then, vector $\mathbf{x}_j$ is presented to the network and the winning neuron r is obtained by finding the most similar codebook entry $\mathbf{m}_r$, e.g., by using Equation 1.

Step 3: The winning codebook vector and its neighbours are updated according to Equation 2. If either $\mathbf{m}$ or $\mathbf{x}$ belong to an unknown class then updating is performed as if they belong to the same class.

Step 4: The coordinates of the winning neuron are passed on to the parent node which updates its vector $\mathbf{c}$ accordingly.

Steps 1 to 4 are executed until a given number of training iterations are performed, or when the network performance has reached a given threshold.

It is not absolutely necessary to execute Step 1 at every iteration. It is sufficient to initialize the class membership of codebook vectors when training starts. Then, a change of class membership can be detected when executing Step 2. Hence, training can be performed by recursively running Steps 2 to 4.

The weight values μ_1 and μ_2 can be computed as they depend on the dimension and magnitude of the elements in $\mathbf{l}$ and $\mathbf{c}$. Given that the Euclidean distance in Equation 1 is

[1]Generally, the neighbourhood radius in SOMs never decreases to zero as otherwise, the algorithm reduces to vector quantization and has no longer topological ordering properties ([1], p. 111)

computed as follows:

$$d = \sqrt{\mu_1 \sum_{i=1}^{p} (\mathbf{l}_i - \mathbf{m}_i)^2 + \mu_2 \sum_{j=1}^{2o} (\mathbf{c}_j - \mathbf{m}_{n+j})^2} \qquad (6)$$

Thus, μ_1 and μ_2 balance the influence of $\mathbf{l}$ and $\mathbf{c}$ to the distance measure. Ideally, the influence of the data label and the coordinate vector on the final result is equal. A way to obtaining the pair of weight values is suggested in [4]:

$$\frac{\mu_1}{\mu_2} = \frac{n}{m} \frac{\sum_{j=1}^{m} (\o(|\mathbf{l}_j|) - \sigma(|\mathbf{l}_j|))}{\sum_{i=1}^{n} (\o(\mathbf{c}_i) - \sigma(\mathbf{c}_i))} \qquad (7)$$

where $\o(|\mathbf{l}_i|)$ is the average absolute value of the i-th element of all data labels in the data set. Similarly, $\o(\mathbf{c}_i)$ is the average of the i-th element of all coordinates. The data label $\mathbf{l}$ is available for all nodes in the data set. The coordinate vector $\o(\mathbf{c}_i)$ needs to be approximated by assuming that the mapping of nodes is at random. Hence, the values in $\o(\mathbf{c}_i)$ are simply half the extension of the map. Moreover, $\sigma(\mathbf{v}_i)$ is the standard deviation of the i-th vector element of a vector $\mathbf{v}$. To obtain unique value pairs we make the assumption that $\mu_1 + \mu_2 = 1$.

In the case where the maximum out-degree of the data is zero, the SOM-SD reduces to the standard SOM model.

We found that in the case that the winning neuron is of a different class as the input vector, then all neurons on the map are affected by the rejection procedure. This adds noise to the learning procedure of remote neurons which may have little relation to the winning neuron. In this paper we propose a rejection strategy which is similar to LVQ [1]. To achieve this, we apply the rejection term only to a local cluster of neurons which is formed by neurons that are of the same class as the winning neuron, and are located closest to it. All other neurons that are not in this cluster are updated as in the unsupervised case. This approach not only eliminates the addition of noise during learning but also accelerates the adaption of remote neurons. This is because remote neurons are drawn towards the input vector even if the vector was classified incorrectly increasing the likelihood that the input vector is mapped away from the winning neuron in subsequent iterations.

IV. EXPERIMENTAL RESULTS

This section investigates the capabilities of the proposed method through a comparison with an unsupervised SOM-SD and a supervised SOM-SD model which applies the rejection term globally [6]. We chose a real world learning problem from the area of pattern recognition where the task is to classify company logos.

The task defined by the logo recognition problem is to recognize and classify company logos such as those shown in Figure 2. This dataset has already been applied to other neural models capable of dealing with structured information such as recursive cascade correlation architecture and recursive multilayer perceptron [7]. The dataset consists of 39 different classes of logos which are available in the form of digital images [2]. There are 300 different samples available for each of the 39 classes, producing a total set of 11700 images.

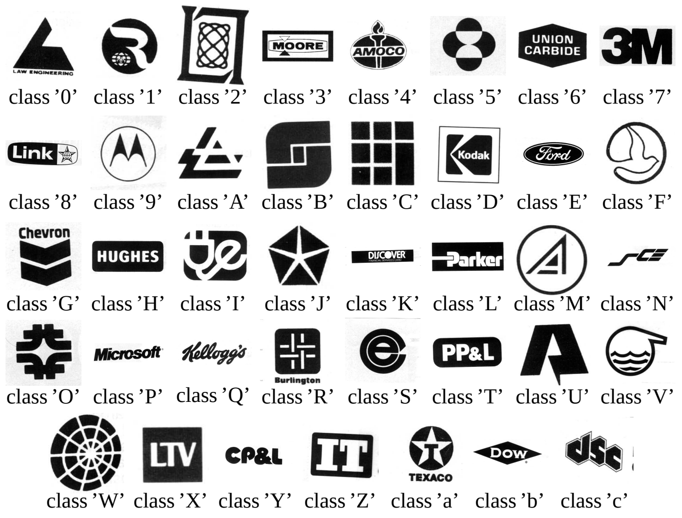


Fig. 2. The original data set of logos. 39 different instances of logos define the 39 classes for the learning problem. In this Figure, the images are scaled to feature the same horizontal extension.

A graph representation is extracted from each of the images by following procedures based on a contour-tree algorithm described in [7]. The result is a training set consisting of 5850 graphs featuring a total of 55547 sub-tree structures or nodes, and a validation data set with 5850 graphs featuring 55654 sub-trees in total. Each node in the graph had a 12-dimensional numeric label attached which consists of:

(1) The area consumed by the contour, which is the number of pixels surrounded by the contour. This value is normalized with respect to the maximum value among all contours.
(2) The outer boundary of the contour in pixels normalized with respect to the largest boundary found in the picture.
(3) The number of pixels found inside the area enclosed by a contour that feature a black color value. The value is normalized with respect to the maximum number of black pixels found in any of the contours of the picture.
(4) The minimum distance in pixels between the image barycenter and the contour. This value is normalized with respect to half of the diagonal of the image bounding box.
(5) The angle between a horizontal line and the line drawn through the image barycenter and the contour barycenter. The angle is quantized in order to reduce the sensitivity to rotation in the image. Eight possible values are coded as real numbers in $[0, 1]$ with a uniform sampling.
(6, 7) The maximum curvature angle for convex sides, and the maximum curvature angle for concave sides.
(8, 9) The number of points for which the curvature angle exceeds a given threshold for convex (1st value) and concave regions (2nd value).
(10) The smallest distance in pixels between the contour and other contours.

[2]The logo dataset was provided by the Document Processing Group, Center for Automation Research, University of Maryland.

(11, 12) The smallest distance between the contour and the two contours with the largest outer boundary.

The root nodes represents an image as a whole, whereas its direct descendants represent outer contours. Recursively, a descendant represents a contour which is located inside the contour represented by its parent node. As a consequence, it is the intermediate and leaf nodes that hold information about the detail of a logo, and the structure of a graph represents the relationship between the components of a logo. The data label assigned to the root node holds little information.

The maximum out-degree is 7. Hence, the dimension of the input vectors and codebook vectors is $12 + 2 * 7 = 26$.

We first determine a good choice of value pairs for the weight values μ_1 and μ_2 since it was reported in [4] that in practice, μ_2 needs to be considerably larger than the theoretical optimum to achieve a good performance when dealing with graphs. To perform this task we trained networks on a range of values for μ_i and used a learning rate $\alpha(0) = 1$, a rejection rate $\beta(0) = 0.05$, a neighbourhood spread $\sigma = 40$, and trained it for 250 iterations. These values were chosen arbitrarily but seemed reasonable. The result is displayed in figure 3.

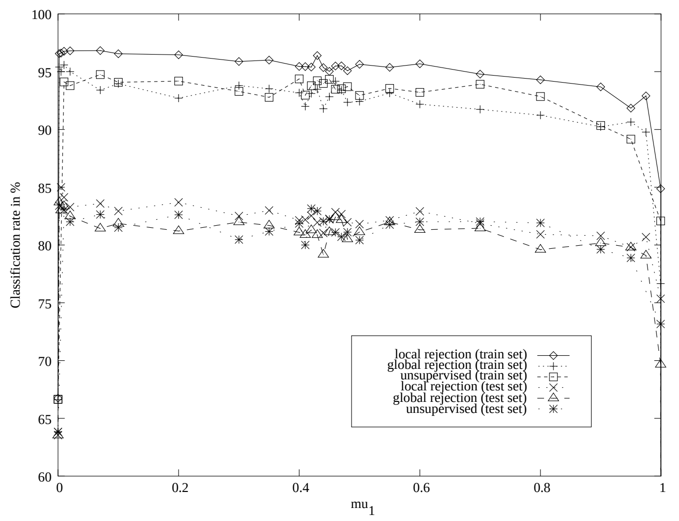


Fig. 3. Dependence of the weight value μ_1 on the classification and generalization performance of a SOM-SD network. Here, $\mu_2 = 1 - \mu_1$.

Equation 7 suggests an optimal value for $\mu_1 \approx 0.999347, \mu_2 = 1 - \mu_1$. However, the experiment confirms that μ_2 needs to be chosen considerably larger to achieve a good performance. Here, we find that $\mu_1 = 0.43$ is best for the unsupervised case, $\mu_1 = 0.01$ is best when engaging rejection globally, and $\mu_1 = 0.07$ when rejecting local clusters only. Nevertheless, the SOM-SD is relatively robust to the choice of the weight values given that the performances change only little over a wide range of μ.

Using the best weight values we then trained the networks with various rejection rate. This experiment will illustrate the effect of rejection on the classification performance. In Figure 4 we show the result for a large range of rejection rates. It is seen that the size of the rejection rate can have a significant impact on the classification performance of the

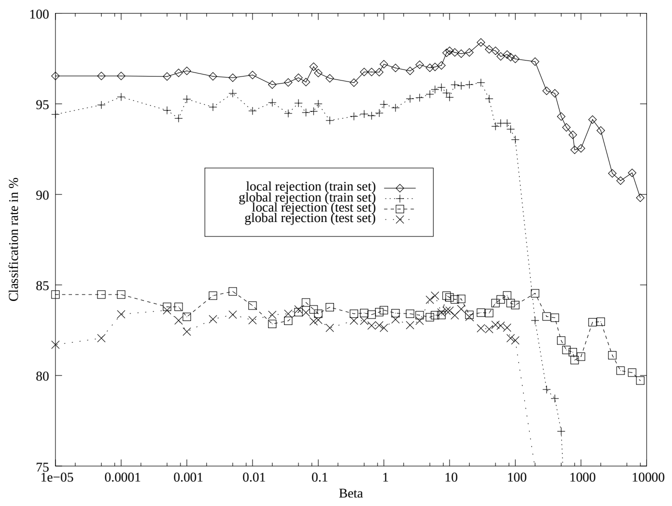


Fig. 4. Performance of the sSOM-SD depending on the rejection rate. Classification performances are shown.

SOM. It is also seen that by restricting the rejection to a local cluster of neurons we generally outperform the global rejection method, and that the local rejection approach is considerably less sensitive to large rejection rates. We find that the best rejection rates are 12 and 15 for the global and local rejection approach respectively. At these rates, the classification performance is 96.05% when rejecting globally and 97.77% when using the local rejection approach. The generalization performance is improved from 83.32% to 84.23%.

Further experiments were conducted to investigate the sensitivity of the networks to the various learning parameters, and to obtain best initial values. Figure 5 trains a number of networks by using the best values for β and μ as determined previously. All other parameters remained unchanged except

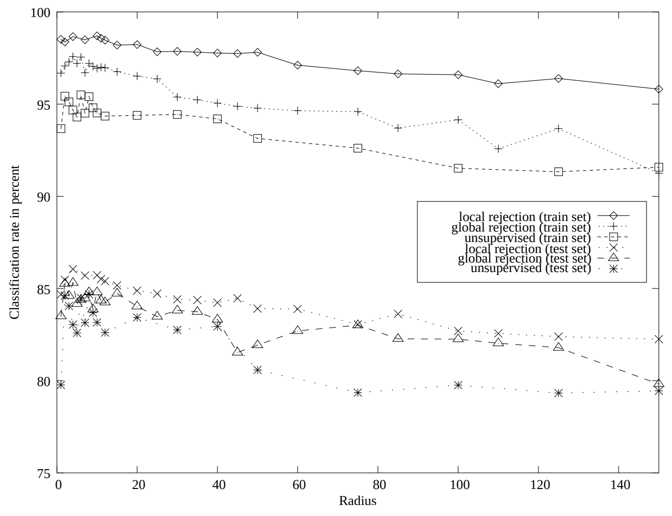


Fig. 5. Performance of the sSOM-SD when varying the initial neighborhood radius $\sigma(0)$. Classification performances are shown.

for the size of the neighborhood radius $\sigma(0)$. It is found that the three methods for training a SOM-SD perform best when trained with a relatively small $\sigma(0)$. This appears to be a common property of SOM-SDs since this has already been observed in [4], [5], [6]. Nevertheless, we find that applying rejection to a local cluster of neurons again outperforms the other two approaches consistently, and that our approach is least sensitive to very small values for $\sigma(0)$. We find that the optimal choice for $\sigma(0)$ is 6 for the unsupervised approach, and 4 for both of the supervised approaches.

Using the best learning parameters as obtained so far, we further investigate the influence of the number of training iterations on the performances of the networks. The experimental results shown in Figure 6 confirm the advantage of

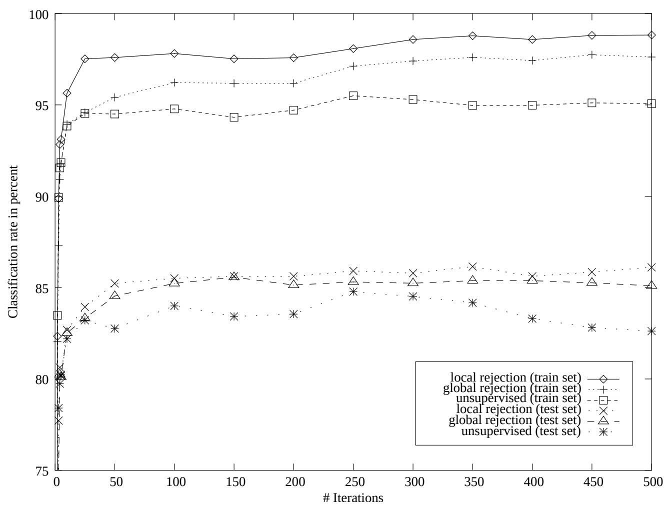


Fig. 6. Performance of the sSOM-SD when varying the number of training iterations. Classifi cation performances are shown in percent.

supervised network training on the classification and generalization performance. Again, the local rejection method consistently outperforms the other methods. It is also seen that our approach produces good results already after as little as 50 training iterations while the other approaches require at least 100 iterations. This confirms a statement made earlier that the local rejection method should accelerate network training.

Further experiments were conducted on initial values for $\alpha(0)$, the network size, and the size of the training set. Though the details of these experiments are omitted we can report that the approach of engaging rejection on local clusters generally produced superior results in both, the classification performances as well as robustness to extreme values of training parameters. The best overall performance achieved with the unsupervised approach was 95.50% classification, and 84.78% generalization. The best supervised network which engaged the rejection term globally achieved 97.60% classification, and 85.38% generalization. Our method of restricting rejection to a local cluster of neurons improved the performance to 98.78% classification, and 86.35% generalization.

Note that in this paper, we have used the classification performance as an indicator on the mapping precision of the SOM. The experiments demonstrated that the mapping precision is influenced positively by utilizing teacher signals if they are available. In addition, the proposed approach extends the capabilities of a SOM towards classification tasks. While SOMs are not typically used for classification tasks it has become possible to compare supervised SOMs with other neural network models which are commonly used for such tasks. Recursive multi-layer perceptron networks (RMLP), and recursive cascade correlation networks (RCC) are examples of neural models which are commonly trained in a supervised fashion, and applied to classification tasks [8]. When applied to the dataset used in this paper, a good RMLP achieves 99.81% classification, and 89.01% generalization performance. Similarly, a good RCC network can achieve 99.98% classification, and 72.94% generalization [8].

V. Conclusions

It has become evident that the use of a rejection term for the purpose of training a SOM in a supervised fashion is an efficient method leading to a significantly improved mapping of data. The idea presented in this paper of restricting supervision to neurons which are nearby the winning neuron and which have a close resemblance to it has lead to a further improvement of the network performance. At the same time, it was found that the robustness of the SOM to learning parameters such as σ, α, β, and μ is improved further. We also rarely observed problems with local minima which may be an indication that the algorithm produces more stable SOMs though this needs further investigation.

The described method of supervision is very general in nature in that it can be employed to a very wide range of flavors of SOM as was demonstrated through the implementation in to SOM-SD. The algorithm also handles missing class information in the data set efficiently in that it falls back to the unsupervised case in those instances.

These improvements come at almost no additional computational cost. A simple `if` statement in the neuron update algorithm is all that is different between the training algorithms of supervised and the unsupervised SOM-SD models.

Note that there is no convergence theorem for this training algorithm introduced in this paper. This deficiency also afflicts the general SOM model.

References

[1] T. Kohonen, *Self-Organizing Maps*, ser. Springer Series in Information Sciences. Berlin, Heidelberg: Springer, 1995, vol. 30.

[2] B. Fritzke, "Supervised learning with growing cell structures," in *NIPS*, Denver, 1993.

[3] M. Varsta, J. Del, R. Millan, and J. Heikkonen, "A recurrent self-organizing map for temporal sequence processing," in *Proc. 7th Int. Conf. Artifi cial Neural Networks, ICANN'97*, 1997, pp. 421–426.

[4] M. Hagenbuchner, A. Sperduti, and A. Tsoi, "A self-organizing map for adaptive processing of structured data," *IEEE Transactions on Neural Networks*, Accepted for publication in 2002.

[5] M.Hagenbuchner, A. Tsoi, and A. Sperduti, "A supervised self-organizing map for structured data," in *Advances in Self-Organising Maps*, N.Allinson, H.Yin, L.Allinson, and J.Slack, Eds., 2001, pp. 21–28.

[6] M. Hagenbuchner and A. Tsoi, "A supervised self-organizing map for structures," in *Special issue PRL 2003*. Elsevier, submitted.

[7] P. Frasconi, E. Francesconi, M. Gori, S. Marinai, J. Sheng, G. Soda, and S. A., "Logo recognition by recursive neural networks," in *Second International Workshop on Graphics Recognition, GREC'97*, R. Kasturi and L. K. Tombre, Eds. Springer-Verlag, 1997, pp. 104–117.

[8] M. Hagenbuchner and A. Tsoi, "Recursive cascade correlation and recursive multilayer perceptron, a comparison," *IEEE Transactions on Neural Networks*, 2002 (Submitted).