On the usefulness of self-organizing maps for the clustering problem in vector quantization

Pasi Fränti

Department of Computer Science,
University of Joensuu, Box 111
FIN-80101 Joensuu, Finland

Abstract

Neural networks have been applied almost everywhere in image analysis, processing and compression. Kohonen's self-organizing feature maps (SOM) offers one approach that has also been applied for the clustering problem. The aim of this study is to find out the suitability of the SOM algorithm for the clustering problem. Specifically we want to find out how the algorithm compare with other clustering algorithms as measured by the quality of the clustering results, but also considering the robustness, reliability and usability of the algorithm.

1. Introduction

Clustering is an unsupervised classification problem where the aim is to partition a set of data vectors into a given number of clusters (classes). The vectors having similar features should be grouped together and vectors having different features to different groups. The vectors can be a training set for a real classification application, data from an image analysis application (e.g. medical images), or image to be quantized or compressed. The correct classification of the training vectors is either not known, or the aim is simply to find, for the given training set, the best possible clustering minimizing a predefined evaluation function (typically mean square error).

Neural networks have been applied almost everywhere in image analysis, processing and compression. Kohonen's self-organizing feature maps (SOM) [10] offers one approach that has also been applied for the clustering problem [1, 13]. It is a rather standardized package and publicly available (ftp://cochlea.hut.fi/ pub/som_pak/), therefore leaving the researcher for solving more important application specific problems such as to study what are the important features to be extracted from the image, how to interpret the result of the classification, or to implement the classifier in a real application such as quality control in computed aided manufacturing.

The aim of this study is to find out the suitability of the SOM algorithm for the clustering problem. Specifically we want to find out how the algorithm compare with other clustering algorithms measured by the quality of the clustering results, but also considering the robustness, reliability and usability of the algorithm. The motivation for this study is that, the SOM algorithm has not been shown to perform significantly better than other clustering algorithms, but on the other hand, people keep saying how well it works on most problems. The most frequently asked question, besides the universal question "how much time it requires?", is probably "have you tried neural networks?".

The use of the SOM algorithm is often argued by the following reasons:

- It performs better than the standard GLA.
- It is less dependent on the initialization.
- It smoothens possible irregularities in the training set.
- The neural network is directly applicable for classification after the learning phase.
- The learning may continue (re-adaptation) when used in the classification.
- The fact that the neighboring vectors in the neuron structure are similar to each other can be exploited.

In the present work we test the above presumptions. The clustering problem is studied as an optimization problem where the aim is to optimize the clustering for the input data. We also study the clustering performance in two cases where the training set is not equal to the original input data to be clustered. In one case the training set is distorted by gaussian random noise, and in another case the vectors are prequantized (from 8 to 5 bits per pixel) before the clustering in order to reduce the amount of calculations. The other potential advantages of SOM are also discussed on the basis of the experiments, and on the knowledge of the other clustering methods and vector quantization.

2. Clustering problem

In vector quantization [8], the aim is to map a set of input vectors into a smaller subset of code vectors (codebook). In data compression, reduction in storage space is achieved by storing the index of the nearest code vector instead of the original data vector. Clustering algorithms are used for optimizing the codebook for a given training set. The training set can be a set of sample vectors, which hopefully corresponds to the actual data to be quantized. In applications such as color image quantization, the codebook (called color palette) is optimized directly for the training set [14].

2.1. Problem formulation

The clustering problem is defined as follows. Given a set of N vectors $X=\{x_1, x_2,\ldots, x_N\}$, partition the data set into M classes such that similar vectors are grouped together and vectors having different features belong to different groups. Partition $P=\{p_1, p_2,\ldots, p_N\}$ defines the classification by giving for each input vectors the partition index (p_i) of the class where it is assigned to. Each class is described by its representative vector $C=\{c_1, c_2,\ldots, c_M\}$.

Each vector x_i has K features (x_i^k). For simplicity, we assume that the features are numerical and have the same scale. If this is not the case they must first be normalized. The distance between two vectors x_1 and x_2 is measured by the *Euclidean distance* and it is calculated as:

$$d(x_1,x_2)=\sqrt{\sum_{i=1}^{K}\left(x_1^i-x_2^i\right)^2} \qquad (1)$$

The quality of the clustering is evaluated by calculating the *sum of squared distances* between the input vectors and their cluster centroid:

$$f(P,C)=\sum_{i=1}^{N}d\left(x_i,c_{p_i}\right)^2 \qquad (2)$$

The cluster centroid c_i is the mean vector of all input vectors in cluster i, and it is calculated as:

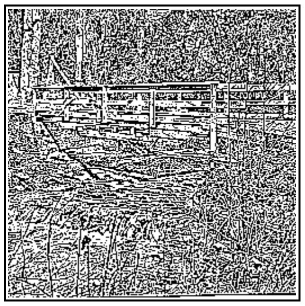
$$c_j=\frac{\sum_{p_i=j}x_i}{\sum_{p_i=j}1},1\le j\le M \qquad (3)$$

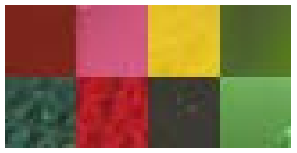
The set of cluster centroids (C) is used as a codebook in the vector quantization. In other clustering applications, the primary goal can be finding the classification with no regard of the actual cluster centroids. In this case the output of the clustering is the set of partition indices (P).

2.2. Test set

We consider the four training sets shown in Fig. 1. *Bridge* consists of 4×4 pixel blocks sampled from a gray-scale image (8 bits per pixel). The data set is very sparse and no clear cluster boundaries can be found. *Bridge-2* has the blocks of *Bridge* after a quantization into two values (0/1) according to the average pixel value of the block, as in *block truncation coding*, see [5]. The vectors in this data set are binary (0/1), which makes it an important special case for the problem.

Bridge
(256×256)
K=16, N=4096


Bridge-2
(256×256)
K=16, N=4096


The dominant colors are:
TOP: *dark red, pink, yellow, dark green,* BOTTOM: *grayish green, red, black, light green*

Blocks
(80×40)
K=2, N=3200*

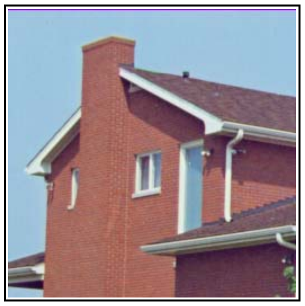

House
(256×256)
K=3, N=65536**

Figure 1. Sources for the data sets. *Only red and green component are used. **The pixels were quantized to 5 bits per pixel. The number of unique colors in *Blocks* is 998, and in *House* 1837.

The third data set (*Block*) is an example of a classification problem. The vectors are from an artificially prepared *RGB* image consisting of samples from 8 different sources. We use only the red and green color components in order to have a two-dimensional data set for illustration purposes, see Fig. 2.

The fourth data set consists of the *RGB* color vectors from the color image *House*. The application of this data set is to generate a color palette for color image quantization. The color resolution is prequantized to 5 bits per pixel before the clustering. The number of unique vectors decreases from 65536 to 1837.

2.3. Generalized Lloyd algorithm

Generalized Lloyd algorithm (GLA), also known as the *LBG* (due to [11]), *K-means* [12], or *C-means algorithm*, is probably the most widely known method. It starts with an initial solution, which is iteratively improved until a local minimum is reached. In the first step, the input vectors are partitioned into a set of *M* classes by mapping each vector to its nearest code vector (cluster centroid):

$$p_i = \arg\min_{1 \le j \le M} d\left(x_i, c_j\right) \tag{4}$$

In the second step, the code vectors are replaced by the centroids of the new clusters, according to (3). The quality of the new solution is always better than or equal to the previous one. The algorithm is iterated as long as improvement is achieved. The number of iterations depends on the data set, and on the quality of the initial solution. Ten to fifty iterations are usually needed when starting from a random initialization.

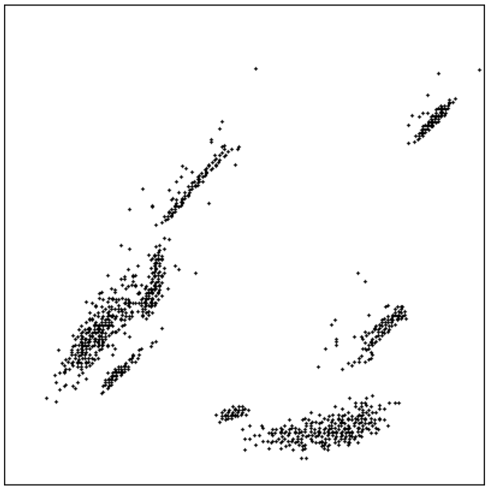


Figure 2. Two dimensional plot of the *RG*-color vectors from *Blocks*.

3. SOM algorithm

A self-organizing feature map (SOM) is composed of a discrete 1-D or 2-D lattice of neurons, each having its own weight vector w_i. During the learning phase, the SOM adapts to training data by self-adjusting the weight values of the neurons. The SOM algorithm can be directly applied to the clustering problem because of similar data structures: the neurons correspond to the code vectors (or cluster centroids) in vector quantization ($w_i = c_i$).

A description of the SOM algorithm is given in Fig. 3. The notations are from the clustering problem. We initialize the neurons by random values. The way of initialization is not important. In the learning phase, training vectors (x_i) are input to the network one at a time. For each training vector, its closest neuron (c_j) is found. The closest neuron and its neighboring neurons are modified by transferring them towards the input vector.

The updated neuron is a weighted average of the original vector (c_i) and the input vector (x_i):

$$c_j \leftarrow c_j + \alpha(t,d) \cdot \left(x_i - c_j\right) \tag{5}$$

where α defines the learning rate. The learning rate gradually decreases with the time (t).

```
SOM-algorithm( X, C ):
    RandomizeNeurons($C$).
    $D \leftarrow D_{max}$.
    $t \leftarrow 0$.
    REPEAT
        REPEAT $T$ times
            LearnVectors($X, C, D, t$).
            $t \leftarrow t + 1$.
        END-REPEAT
        $D \leftarrow D - 1$.
    UNTIL $D < 0$.
END-SOM

LearnVectors( X, C, D, t ):
    FOR each $x_i \in X$ DO
        $j \leftarrow$ FindNearestNeuron($x_i, C$).
        FOR $d = -D$ to $D$ DO
            $p \leftarrow j+d$
            $c_p \leftarrow c_p + \alpha(t,d) \cdot (x_i - c_p)$
        END-FOR
    END-FOR
END
```

Figure 3. Description of the SOM algorithm.

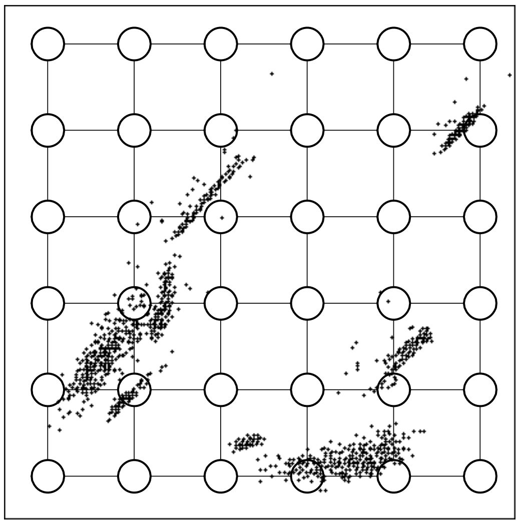


Figure 4. Illustration of the structure of a 2-D SOM.

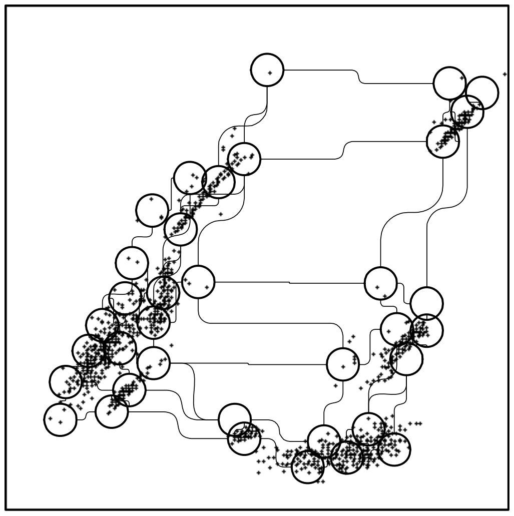


Figure 5. Illustration of the 2-D SOM after training.

The size of the neighborhood (*D*) determines how many of the neighboring neurons are updated. It also decreases with the time, and in the last iteration only the matching neuron is updated (*D*=0). With each neighborhood size we iterate the training set *T* times.

The SOM algorithm is similar to the GLA in two ways: they are both iterative, and they both make similar updates to the current solution. The difference is the way the updates are made in practice. The GLA calculates the new centroids as the exact averages of all vectors in each cluster after every training vector is partitioned. The SOM algorithm, on the other hand, uses a competitive learning technique where each training vector affects its nearest code vector immediately when it is processed.

3.1. Network topology

The vectors in SOM are connected with each other, usually by a symmetric 1-D or 2-D structure. A 2-D structure is often chosen for illustration purposes. The data, on the other hand, is usually multi-dimensional but it does not imply that a multi-dimensional network structure should be used. On the contrary, too many connections may restrict the movement of the neurons. The main effect of the structure is that neighboring neurons represent similar vectors to each other.

The main purpose of the network is that the input of a training vector affects several neighboring code vectors. The *fuzzy GLA* [3] has similar approach by distributing the mapping of a vector to several neighboring vectors. The difference is that the neighboring vectors are adaptively found as the nearest code vectors, whereas SOM uses a fixed neighborhood structure. See Fig. 4 and 5 for an illustration of the network structure for the sample data set.

3.2. Learning rate

The learning rate α depends on the time (iteration count). It is monotonically decreasing with time and belongs to the range $0 \le \alpha \le 1$. The learning rate depends also on the distance (*d*) of the processed neuron from the matching neuron (see Fig. 6). In a linear model, α is inversely proportional to the number of iterations (*t*) performed:

$$\alpha(t,d) = A \cdot \frac{T-t}{T} \cdot e^{-d/D} \qquad (6)$$

Here *A* is a constant giving the maximum amount of change. The last part of the equation weights the learning rate according to the location of the neuron. The weight is 1 for the matching neuron and less for the other neurons, see Fig. 7. In an exponential model, the learning rate is defined as:

$$\alpha(t,d) = A \cdot e^{-t/T} \cdot e^{-d/D} \qquad (7)$$

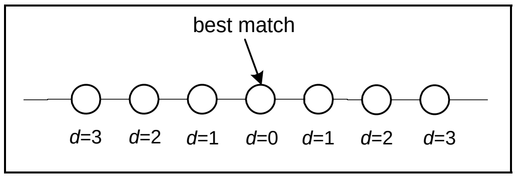


Figure 6. Illustration of the neighborhood of 1-D SOM.

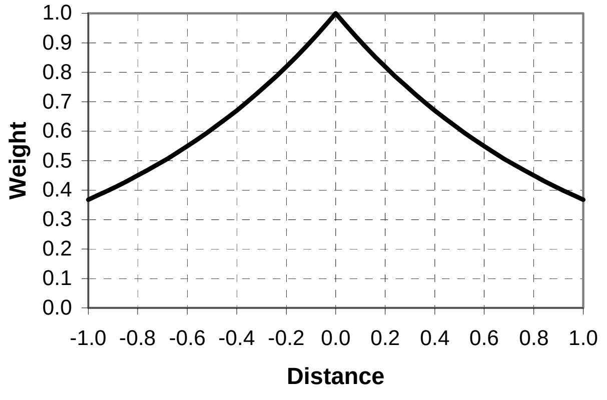


Figure 7. Weighting of the updates in the neighborhood.

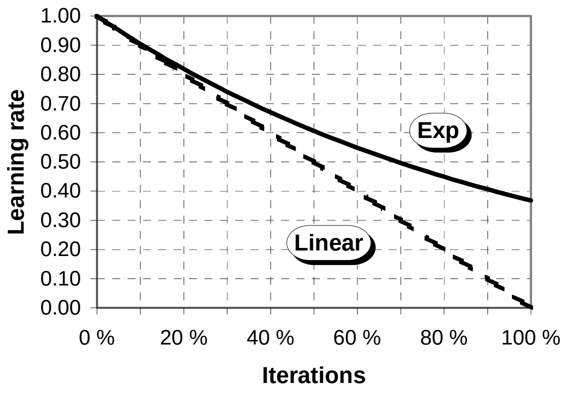


Figure 8. Behavior of the two learning rate functions.

The behavior of the two learning rate functions is illustrated in Fig. 8. In practice, any monotonically decreasing function is suitable. The performance of the algorithm is found to be more dependent on the choice of the constant A than on the choice of the learning rate function.

3.3. Parameter setup

The SOM algorithm includes several parameters to set up: the size of the initial neighborhood $D_{\max}$, the number of iterations T, and the maximum learning rate A. There is no easy way to find the optimal set up because the parameters depend on each other, and also on the input data.

The number of iterations (T) is probably the easiest to set up. It should be as high as computation time can be spent. The convergence of SOM is rather slow and therefore a high number of iterations is needed, approximately about 100-1000 iterations at minimum. It is noted that the choice of T affects the optimal choice for the other parameters.

The size of the initial neighborhood ($D_{\max}$) should be somewhere between 1 to M, where M is the number of clusters. Most likely closer to 1 than to M, because with a very large neighborhood the freedom of the neuron movements will be reduced too much. The selection of D=0, on the other hand, would loose the effect on the neighborhood completely and the algorithm would then be a simple competitive learning.

The maximum learning rate could be set to A=1 because the learning rate can vary in the full range from 1.0 to 0.0. However, the learning rate values close to 1 have practically random effect on the solution and are therefore meaningless. On the other hand, the learning rate should be able to have very small values (close to 0) so that fine-tuning may happen in the solution. It is therefore better to have a smaller A than a large one. This is especially true if the number of iterations is small.

Most significant improvement in the clustering appear in the later stages of the algorithm when the neighborhood size is very small, or shrunk to zero. It is therefore most critical for the learning that the parameter setup is correct in the last stages of the algorithm (when $D \le 2$). The parameter A should be high enough so that the learning rate does not become negligible before the algorithm reaches these critical stages. This is a potential danger especially if $D_{\max}$ is very large.

The optimal choice of the parameters A and $D_{\max}$ clearly depends on each other, and on the choice of the number of iterations T.

3.4. Adaptive number of iterations

A rather straightforward improvement on the SOM algorithm can be concluded from the previous discussion. Since the later stages of the algorithm are the most vital, the largest number of iterations, and the highest variation in the learning rate should appear then.

We propose the use of a variable number of iterations, which is a function of the neighborhood size (D). Given a maximum number of iterations $T_{\max}$ (applied when D=0), the current number of iterations is:

$$T(D) = \left(\frac{1}{2}\right)^{D} \cdot T_{\max} \tag{8}$$

The number of iterations progressively increases (doubles) after each time the neighborhood size is shrunk. In this way, the algorithm iterates less in the early stages and more in the later stages. This is also reasoned by the fact that the learning is more stochastic when the neighborhood size is large and, on the other hand, it has maximum variations at the critical stage. The total number of iterations is:

$$T_{\text{total}} = \sum_{D=0}^{D_{\max}} \left(\frac{1}{2}\right)^{D} \cdot T_{\max} \approx 2 \cdot T_{\max} \tag{9}$$

Having the same number of iterations in the last stage as in the original algorithm ($T_{\max}=T$), we also achieve a significant saving in running time ($2 \cdot T < D_{\max} \cdot T$) assuming that the size of the initial neighborhood $D_{\max}>2$.

Note that $T_{\max}$ should be given as a power of 2 if the approximation of (9) is used when applied in the learning rate (6). It is though possible to use any value of $T_{\max}$. For example, given $D_{\max}=10$ and $T_{\max}=100$, we get the values $T_i = \{1, 1, 1, 1, 2, 3, 6, 13, 25, 50, 100\}$, where $0 \le i \le 10$.

4. Experiments

We study the performance of SOM for the data sets of Fig. 1 using 8 clusters for *Blocks*, and 256 for the rest. We apply a 1-D network. The program includes all the options discussed in the previous sections. We use the default parameter setup as A=0.1, $D_{\max}$=10, and $T_{\max}$=1000.

The results for *Bridge* in Fig. 9 illustrates that the optimal choice for $D_{\max}$ depends on the total number of iterations ($D_{\max} \cdot T$). The effective number of iterations (T) is adjusted so that the total number of iterations is always fixed. We can tentatively say that the use of a large total number of iterations implies the use of a small neighborhood size ($D_{\max}$), and vice versa.

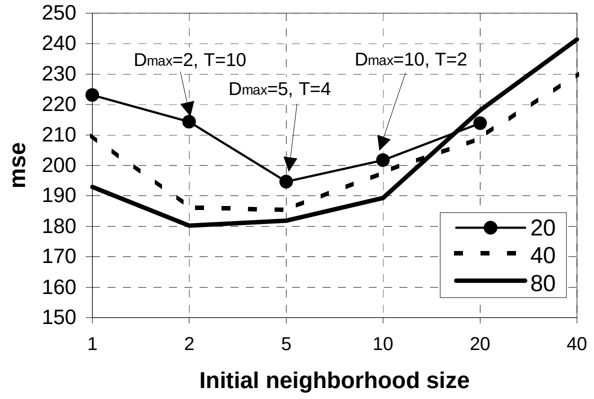


Figure 9. Clustering performance (for *Bridge*) by fixing the total number of iterations ($D_{\max} \cdot T$) to 20, 40 and 80.

The results for the smaller set *Blocks*, on the other hand, give contradicting results, see Fig. 10. For this data set we should always use the maximum neighborhood size. The experiments also show that better results are achieved using the adaptive number of iterations. The optimal choice of the maximum learning rate (A) for *Blocks* is also much smaller than for *Bridge*.

The results of Fig. 11 demonstrates the importance of correct parameter setup. It is very easy to set the parameters to get very bad results but rather difficult to adjust the parameters to find the optimal setup. The use of the linear function for the learning rate was more robust than the exponential function.

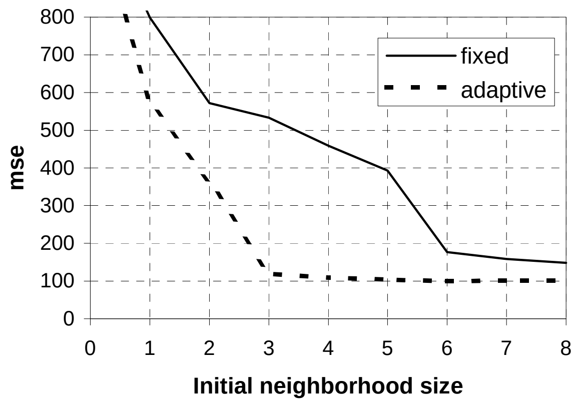


Figure 10. Clustering performance (for *Blocks*) by fixing the total number of iterations ($D_{\max} \cdot T$) to 1000.

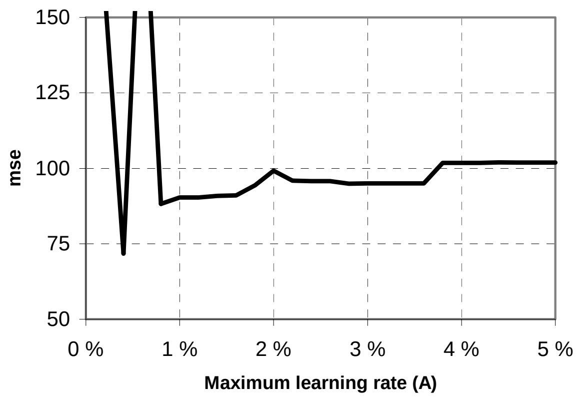


Figure 11. Optimal choice of A for *Blocks*.

Table 1. Clustering performance inside training set.

	Bridge	***Bridge-2***	***Block***
Random	251.32	1.51	265.52
GLA	179.68	1.48	60.46
PNN	169.15	1.44	57.82
SOM	173.63	1.40	71.80
GA	**162.09**	**1.28**	**55.18**

Table 2. Clustering performance outside training set.

	Block		***House***	
	distorted	original	quantized	original
Random	211.74	204.57	6.88	10.99
GLA	94.20	**82.33**	4.80	9.45
PNN	95.67	93.13	4.38	8.91
SOM	94.25	87.10	N / A	N / A
GA	**94.18**	84.27	**4.06**	**8.89**

The results for the best parameter combination are next compared with other clustering algorithms. We use the following methods as the point of comparison:

- *Random*
- *Generalized Lloyd algorithm* (*GLA*) [11]
- *Pairwise Nearest Neighbor* (*PNN*) [4]
- *Genetic algorithm* (*GA*) [7]

Random solution is generated by selecting *M* randomly chosen training vectors as the code vectors, and by mapping all other vectors to the nearest code vector according to (1). For the *PNN* we use the implementation introduced in [6].

The results are summarized in Table 1. It is shown the GA gives best results in all cases, whereas SOM results are sometimes better than that of the GLA but for the *Blocks* it is worse. It is noted that the optimal set of parameter combination was very difficult to find out.

The performance of the clustering algorithms outside the training set is shown in Table 2. In the first case (*Blocks*), the training set is distorted by gaussian random noise, and in the second case the vectors are prequantized from 8 to 5 bits per pixel before the clustering process.

The results show that the choice of the clustering algorithm is less critical when the training set does not match the input data. No algorithm is clearly better than another. The results also demonstrate that the proper choice of the training set is vital. It is noted that we did not find a proper parameter setup for SOM in the case of *House*. The problem seems to be that after prequantization the frequencies of the training vectors varies greatly. This is not taken properly account by SOM, which performs blind adjustment of the neurons depending only on the learning rate.

5. Discussion

The first hypothesis that SOM has better learning performance than the GLA was verified only in two cases out of three. It is, however, not difficult to outperform GLA at all but practically every method gives better result than GLA. Of the tested methods, the GA was the best in all cases.

The second hypothesis that SOM is less dependent on the initialization is partly true. Unlike in the GLA, the initialization has practically no effect on the final result if the maximum learning rate is setup high enough. On the other hand, the method is extremely sensitive to the parameter setup, and therefore we could say that the dependence on the initialization is changed to a more severe dependence on the correct parameter setup.

The third hypothesis was that the network structure smoothness the result, and might perform better when applied outside the training set. However, there was no such evidence that SOM is better in this sense but the choice of a proper training set itself seems to vital. This is also verified by the studies in [2, 9], which have shown that a very small number of samples are usually sufficient for training purposes.

The fourth hypothesis is that neural network is directly applicable for the classification process. This is true but the classification is trivial process anyway once the distance function is defined, and the learning phase performed. Neural networks does not offer any better data structure than the conventional ones.

The fifth hypothesis stated that SOM allows the learning to continue during the classification. This is true but there is no reason why another classifier could not be re-adapted when more training samples are available. Using a proper history buffer, any classifier can be re-adapted on the fly. However, the question of how this is done is usually not issued as it is implicitly taken account in SOM. In this sense SOM is better because it has one problem less to be solved.

The final hypothesis was the fact that the similarity of the neighboring vectors can be exploited, for example in image compression. This might be true but predictive techniques are commonly used in data compression anyway. It is unlikely that the fixed neighborhood structure offers as good prediction as the ones used in data compression. For example, the code vectors could always be sorted according to the principal axis of the vector space. Therefore SOM does not offer anything more than could be done by other means.

6. Conclusions

As an optimizer we conclude that SOM is clearly not the best choice. The method is only slightly better than the standard GLA but worse than the best methods. As a classifier it neither had anything more to offer than the other clustering methods because the classifier is either trivial to construct, or it is designed in the learning algorithm already.

One benefit of SOM is the possibility of re-adaptation after the training phase, which is implicitly built in SOM due to the competitive learning approach. The connectivity of the clusters in SOM might also prevent single isolated clusters to be created far from the central areas. This could be a benefit in image processing applications but it is more likely a restriction that disallows the best optimization.

Overall, the network structure is a restriction because many good ideas in other clustering algorithms cannot be naturally fitted in SOM without external data structures. The data structure itself is good because it allows global changes in the clustering structure, but the competitive learning approach does not support this capability at all, but all modifications appear via local changes only.

The SOM algorithm is also very slow, and it requires a large number of iterations. The dependence on the initialization is not a problem but instead the method depends on the parameter setup. The main benefit of SOM therefore seems to be limited to the fact that it is more or less a standardized package for the training phase.

Acknowledgments

The work of Pasi Fränti was financed by a grant from the Academy of Finland.

References

[1] T.-D. Chiueh, T.-T. Tang, L.-G. Chen, "Vector quantization using tree-structured self-organizing feature maps", *IEEE Journal on Selected Areas in Communications*, **12** (9), 1594-1599, December 1994.

[2] D.C. Cohn, E.A. Riskin and R.L. Ladner, "Theory and practice of vector quantizers trained on small training sets", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16** (1), 54-65, January 1994.

[3] V. Delport and D. Liesch, "Fuzzy-c-mean algorithm for codebook design in vector quantisation", *Electronics Letters*, **30** (13), 1025-1026, June 1994.

[4] W.H. Equitz, "A new vector quantization clustering algorithm", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **37** (10), 1568-1575, October 1989.

[5] P. Fränti, T. Kaukoranta and O. Nevalainen, "On the design of a hierarchical BTC-VQ compression system", *Signal Processing: Image Communication*, **8** (11), 551-562, 1996.

[6] P. Fränti and T. Kaukoranta, "Fast implementation of the optimal PNN method", *Proc. IEEE Int. Conf. on Image Processing* (*ICIP'98*), Chicago, Illinois, USA, 1998.

[7] P. Fränti, J. Kivijärvi, T. Kaukoranta and O. Nevalainen, "Genetic algorithms for large scale clustering problems", *The Computer Journal*, **40** (9), 547-554, 1997.

[8] A. Gersho and R.M. Gray, *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Dordrecht, 1992.

[9] D.S. Kim T. Kim and S.U. Lee, "On testing trained vector quantizer codebooks", *IEEE Transactions on Image Processing*, **6** (3), 398-406, March 1997.

[10] T. Kohonen, *Self-Organization and Associative Memory*. Springer-Verlag, New York, 1988.

[11] Y. Linde, A. Buzo and R.M. Gray, "An algorithm for vector quantizer design". *IEEE Transactions on Communications*, **28** (1), 84-95, January 1980.

[12] J.B. McQueen, "Some methods of classification and analysis of multivariate observations", *Proc. 5th Berkeley Symp. Mathemat. Statist. Probability* **1**, 281-296. Univ. of California, Berkeley, USA, 1967.

[13] N.M. Nasrabadi and Y. Feng, "Vector quantization of images based upon the Kohonen self-organization feature maps", *Neural Networks*, **1** (1), 518, 1988.

[14] M.T. Orchard and C.A. Bouman, "Color quantization of images". *IEEE Transactions on Signal Processing*, **39** (12), 2677-2690, December 1991.