Title:

Self-Organizing Hierarchical Feature Maps

Authors:

PASI KOIKKALAINEN & ERKKI OJA

Published in:

IJCNN, INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS San Diego, California, June 17-21, 1990.
pp. II-279-II-284

Publications from

THE
INFORMATION PROCESSING LABORATORY

Lappeenranta University of Technology

P.O.BOX 20, SF-53851 Lappeenranta
Finland

Self-Organizing Hierarchical Feature Maps

Pasi Koikkalainen and Erkki Oja
Lappeenranta University of Technology
Department of Information Technology
P.O.Box 20, 53851 Lappeenranta, Finland
e-mail:koikkalainen@ltkka.lut.fi (INTERNET)

ABSTRACT

The topological feature map (TFM) algorithm introduced by Kohonen implements two important properties: a vector quantization (VQ) and a topology preserving mapping. A tree-structured TFM (TSTFM) is presented as a computationally inexpensive alternative to the TFM algorithm. The computational complexity of the TSFTM is $O(\mathrm{logN})$ rather than $O(N)$ for the TFM. In addition, TSTFM has some new properties that prove to be useful for VQ and in the context of visual perception: increased performance in VQ compared to the tree-structured VQ (TSVQ) by Buzo et al, and hierarchical mapping of code vectors.

1. Introduction

In the context of information theory Kohonen's Topological feature map (TFM) algorithm is an extension of the generalized Lloyd algorithm (for Lloyd algorithm see [Lloyd-82]) that is better known as the Linde-Buzo-Gray vector quantization algorithm (LBG) [Linde-80]. In vector quantization (VQ) an n-dimensional vector in Euclidean space is approximated by its closest (minimum distortion) representative among a finite set of code vectors (reference vectors). The most popular algorithm for such a mapping is the LBG algorithm, that produces a locally optimal codebook from training data. The locality is better explained in neural networks as local minima where LBG algorithm may be trapped during the training procedure.

In the domain of neural networks, TFM is one of the best known algorithms for VQ. Both LBG and TFM determine the closest code vector by computing the distortion between the source vector x and each of the code vectors x'_i, the most popular distortion measure being the Euclidean distance $d_E(x,x') = \sqrt{\|x-x'\|^2}$.

The number of code vectors (neurons in Kohonen's algorithm) needed for VQ increases exponentially with vector dimension and so does the computational load. Therefore a lot of research has been done to reduce the search complexity. The full-search algorithm has a complexity $O(N)$, where N is the number of code vectors and both in LBG and in TFM the search time will become dominant with large N. An algorithm for search in a logarithmic time complexity $O(\log N)$ is obtained if the code vectors are arranged in a tree structure [Friedman-77]. This has been implemented in a tree-structured VQ (TSVQ) that was introduced in [Buzo-80]. In general, TSVQ produces a quantizer whose performance is worse than that of full-search VQ, the effect of quantization errors in parent nodes being accumulated to leaf nodes. The result above is for the fixed rate TSVQ that is based on trees of fixed length. Only very recent results indicate that variable-rate TSVQ or

This work was supported by Tekes under Esprit basic research project on subsymbolic computing and analogical modelling

dynamically expanding TSVQ can outperform both the fixed rate TSVQ and the full-search VQ [Chou-89].

This paper presents a tree-structured version of the TFM algorithm (TSTFM) that like the TSVQ reduces the complexity of the original full-search algorithm logarithmically. In some applications TSTFM has an additional benefit over TFM, since it produces a hierarchical mapping of code vectors. This is believed to be important for many problems in vision, cognition, and AI.

2. Neural Networks and Generalization

Many neural network algorithms learn how to generalize their input or training patterns. This is believed to be one of the most important functions of neural models. But what kind of generalizations are useful, would naturally depend on the application where neural networks are used. In the following some remarks are made when the given task is the classification of data patterns.

In general, different classes are separated by creating discriminant surfaces in input space. Such surfaces form decision regions which should correctly classify the given training patterns and with some confidence also classify correctly some unknown patterns of data.

In supervised applications the generalization can be measured by using separate training and test sets: the less there are missclassifications for test set patterns, the better is the generalization for that class. The statistically optimal decision surface for given density functions of input data is Bayesian. In practice the a priori statistics are unknown and the optimal rule can not be obtained.

In unsupervised models the question of generality is more difficult. If these models are used for feature selection then some criteria for faithful generalizations can be found: the model should reduce the dimension of the data or discretize it with minimal loss of information. Unfortunately, such a criterion is difficult to evaluate because the usefulness of the information is very application dependent (for a recent discussion see [Linsker-90]). In current neural clustering and vector quantization algorithms the amount of information reduction is usually defined by a fixed number of neurons in the network. The decision of the proper resolution is difficult to make: in coarse resolution the features are more general and in fine resolution the danger of data loss is minimized. Thus it might prove to be useful to use the multi-resolution principle for feature selection. Using several neural networks with different output resolutions, different levels of generalization would be obtained. The algorithms presented in this paper implement a way to meet these requirements of generality.

3. A simplified TSVQ algorithm

In this section a simplified TSVQ algorithm is briefly reviewed. This algorithm is then extended (in section 4.) with the topological mapping resulting in the TSTFM algorithm. In TSVQ the code vectors are arranged in a tree structure that may be static or dynamic. The static hierarchies are fixed in depth and in breath. In Fig. 1a a static binary tree hierarchy for four leaf nodes is shown. The dynamic hierarchy may expand in depth or in breath according to a certain criterion (Fig. 1b), that could be expressed in terms of similarity measures. Training will take place first at the higher levels of the hierarchy. When the organization for a code vector (neural unit) at any hierarchy level is finished, the plasticity of the unit is frozen and the procedure will be applied for the units in the subsequent layers. The TSVQ algorithm is similar to the recently introduced "Neural Tree Structured Vector Quantization" [Wan-90].

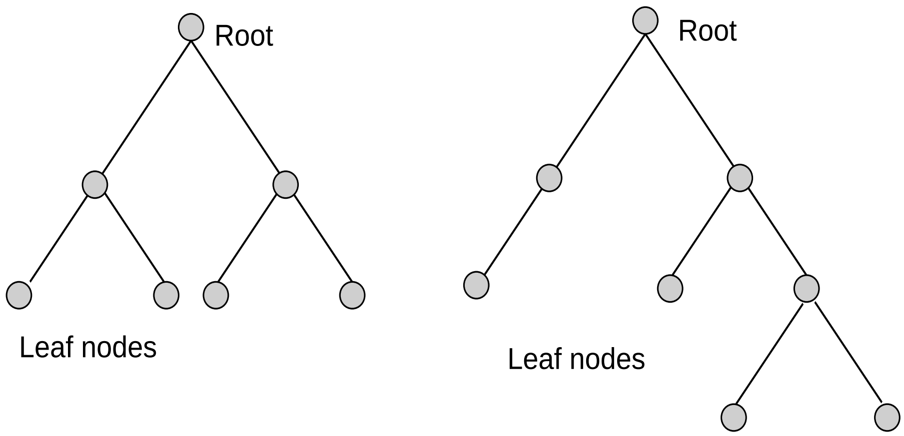

Figure 1. (a) A simple static binary-tree hierarchy. (b) An illustration of a possible hierarchy after organization when a dynamic model is used.

A tree-structured search for the nearest neighbor is done by using the algorithm **Find.Best(**$best$,x**)**, where $best$ is an index of a vector in the set of all code vectors and x is the input sample vector. The search algorithm can be defined by simple recursion:

Find.Best($best$,x**)**
begin
  If node $best$ is frozen **then**
    find a subsequent node son of the node $best$ that satisfies
      $minimize \quad S(son) = d(x'_{son}, x)$;
    make a recursive call **Find.Best(**son,x**)**
  else return the index $best$ and exit
end.

At the initial stage $best$ is the index of the root node.

For simplicity, this TSVQ algorithm is for fixed trees only. For a given input pattern $x(t) \in [x_1, x_2, \ldots x_n]$ at the global iteration number t, a best matching unit index $best$ is searched from the hierarchy tree using tree-search methods. The search criteria can be given, for instance, in terms of similarity between the input pattern x and the code vector (weight vector,synapse vector) x'_j of the unit j: $\|x - x'_{best}\| = \min_j\{\|x - x'_j\|\}$. When found, the code vector x_{best} of the best matching unit is moved towards the input pattern x. Finally the convergence of the best matching unit, the criteria for which may differ within applications, is tested. The algorithm is given briefly below:

THE ALGORITHM FOR SELF-ORGANIZING HIERARCHIES
begin
(initial values and constants:)
    top (index of the unit highest in the hierarchy)
    $x'_{top} = x'_{initial}$;
repeat
1. get new input pattern x;
2. (find best matching unit using the algorithm **Find.Best**:)
    best=*top* (initially the best unit is *top*);
    Find.Best(*best*,x**)**;
3. (Update the best matching unit:)
    $x'_{best} = x'_{best} + \alpha(x - x'_{best})$;
    $iter[best] = iter[best] + 1$ (update the convergence test rule);
4. (Check the convergence for the updated unit:)
    if $iter[best] \geq maxiter$ **then**
        $convergence[best] =$ **true** (unit is frozen);
        for every unit j in the next hierarchy level below *best*
            $x'_j = x'_{best}$ (initialize next level units)
until every unit is frozen
end.

4. The TSTFM algorithm

In this section we shall derive an extension of the TSVQ algorithm outlined in section 3. We shall call this algorithm the TSTFM algorithm as it is a tree-structured search version of the well known TFM algorithm. Both TSVQ and TFM algorithms are so-called structured vector quantizers (for more information see [Gersho-82]). In TFM the structure is in the lateral connections between code vectors, while the structure of the TSVQ has been discussed above. As a combination of these structures, a pyramid-structured VQ is obtained. This structure is used in TSTFM (Fig. 2).

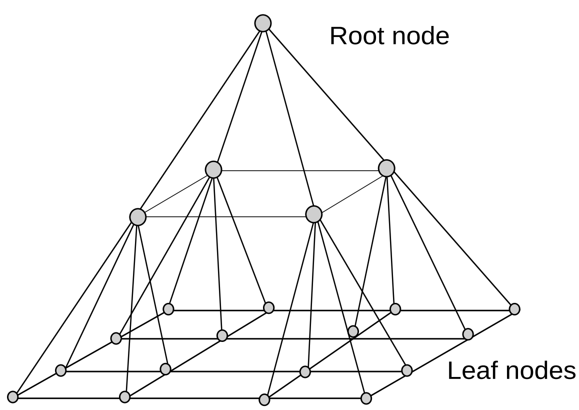

Figure 2. The structure of the TSTFM

The training algorithm is in essential parts the same as in the TSVQ algorithm. The differences are in the search algorithm and in the update rule. The search is done by a tree-search as before but at each step (each node in the tree during search) a lateral search in the direction of lateral neighbours (see [Lampinen-89]) is done to find the locally best code vector index. This lateral search is quite inexpensive (actually only a few nearest neighbors have to be examined) but still vital for the algorithm as explained below. The update rule for the best matching unit is taken from the standard TFM so that the nearest lateral neighbours are updated as well. This may move some of the units over the decision regions of parent nodes in the tree layers above. To prove that the best matching unit has been actually found, a lateral search is executed. This also makes TSTFM tolerant to quantization errors in the parent nodes. Note that the search is always limited to the nodes of the neighbouring parent node on the lattice, since the nodes cannot move too far from their parents. This results in computational complexity $O(\log N)$ that is the same as for TSVQ.

5. Discriminant surfaces

It is interesting to compare how the best matching unit is selected in the input space by TSVQ and TSTFM algorithms. It is generally known that in full search algorithms the decision surface is a set of convex voronoi regions around the code vectors. The decision surfaces below are computed by the algorithms described above for a uniformly distributed data on a two dimensional space.

In TSVQ the space is separated at each level of the decision tree by k-voronoi regions, where k (in this case 4) is the dimension of the tree. At the subsequent level these regions are themself separated by a similar manner. The higher level partitions can be seen also in the decision surfaces of the leaf nodes (Fig 3a).

The combined tree-lateral search in TSTFM results in decision surfaces similar to the regular lattice of the feature map. This produces a uniform pattern of decision regions in the leaf nodes (Fig 3b.). These partitions in the higher levels of the hierarchy follow the same principle but are more coarse with resolution.

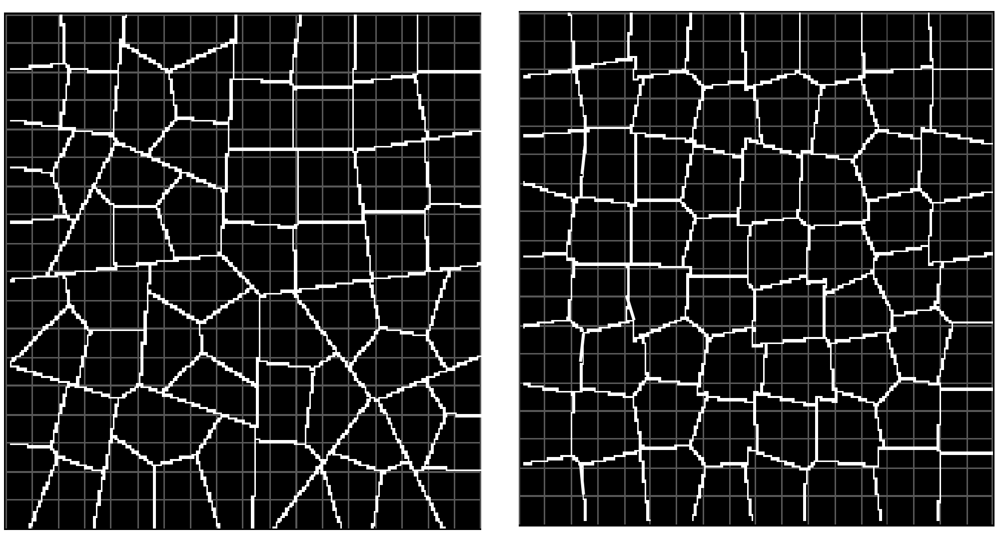

Figure 3. Decision regions of (a) TSVQ (b) TSTFM.

6. Quantization examples

Finally some preliminary quantization results are given for simple artificial data. In the table below a squared average quantization error is computed using uniform and two-class gaussian distributed random samples in two, four, and ten dimensional spaces. In the case of gaussian data the standard deviation was 0.2 in each direction and the class centroids were in 0.3 and 0.7 in each direction. The uniform data was contained in a unit hypercube. The plasticity coefficient α was in both algorithms constant 0.05 and the topological neighbourhood in TSTFM were the nearest neighbours in the lattice. The adaption of neighbours was totally frozen after ten best matches. The convergence criterium for a unit was achieving 200 best matches (in the algorithm $maxiter = 200$). The tree dimension was 4 (the parent unit has four sons) and 4 level trees were used.

dimension	TSVQ	TSTFM	data type
2-D	0.004097	0.003892	gaussian
4-D	0.045873	0.043826	gaussian
10-D	0.267595	0.263861	gaussian
2-D	0.002933	0.002705	uniform
4-D	0.049366	0.050033	uniform
10-D	0.462961	0.460992	uniform

These results indicate that in some applications TSTFM can outperfom TSVQ. It is supposed, that this results from the more uniform structure of the decision surfaces in TSFTM. Note that this data was "easier" than in most practical applications. If the application is speech coding or image quantization the data distribution is not uniform or gaussian.

7. Conclusions

In this paper a tree-structured topological feature map algorithm (TSTFM) was introduced as a computationally inexpensive alternative to the full search TFM. It implements both the topological preserving mapping of the TFM and a logarithmic speedup in quantization of the TSVQ. In addition, TSFTM implements some new properties that are useful in the context of visual perception and cognition: multi-resolution features and hierarchical mapping of code vectors.

References

[Buzo-80] Buzo, A., Gray, A.H.,Jr,Gray, R.M., Markel, J.D., Speech coding based upon vector quantization, IEEE Trans. ASSP, vol 28, no 5., Oct 1980, pp.562-574.

[Chou-89] Chou, P.A., Lookabaugh, T., Gray, R.M., Optimal prunning with applications to tree-structured source coding and modelling, IEEE Trans. Inform. Theory, vol. 35, no. 2, March 1980, pp. 299-315.

[Friedman-77] Friedman, J.H., Bentley, J.L., Finkel, R.A., An algorithm for finding best matches in logarithmic time, ACM Trans. Math. Software, vol 3., Sept 1977, pp. 209-266.

[Gersho-82] Gersho, A., On the structure of vector quantizers, IEEE Trans. Inform. Theory, vol it-28, no. 2, March 1982, pp. 157-166.

[Linsker-90] Linsker, R., Designing a sensory processing system: what can be learned from Principal Component Analysis ?, Proc. IJCNN-90, International Conference on Neural Networks, Washington D.C., Jan 15-19, 1990, pp. II-291 - II-296.

[Lloyd-82] Lloyd S.P., Least squares quantization in PCM, IEEE Trans. Inform. Theory, vol it-28, no. 2, March 1982 pp.129-137.

[Kohonen-82] Kohonen, T., Self-organized formation of topologically correct feature maps, Biological Cybernetics 43, 1982, pp. 59-69.

[Lampinen-89] Lampinen, J., Oja, E., Fast self-organization by the Probing Algorithm, Proc. IJCNN International Conference on Neural Networks, Washington D.C., June 18-22, 1989, pp. II-503 - II-507.

[Wan-90] Wan, E., Ning P., Widrow, B., Neural tree structured vector quantization, Proc. IJCNN-90, International Conference on Neural Networks, Washington D.C., Jan 15-19, 1990, pp. II-267 - II-270.