9 Speedup of SOM Computation

Teuvo Kohonen

9.1 Addressing Old Winners

If there are M map units (neurons) in the SOM, and for a certain statistical accuracy one stipulates that the number of updating operations per unit should be some constant (say, on the order of 100), then the total number of comparison operations to be performed by exhaustive search of the winners is $\sim M^2$.

Koikkalainen [1,2] has recently suggested a speedup method in which a search-tree structure, except its last layer, is replaced by pointers from data items to the next-to-last layer. A more accurate search is then made among the last branches of the tree. We will show below, however, that this idea is not restricted to tree structures, but can readily be added to any SOM software package. The total number of comparison operations can be made $\sim M$, provided that the training vectors have been given in the beginning, i.e., their set is finite and closed.

Assume that we are somewhere in the middle of the training process, whereby the last winner corresponding to each training vector has been determined; then the training vectors can be expressed as a linear table, with a *pointer* to the corresponding *tentative winner location* stored with each training vector (Fig. 11).

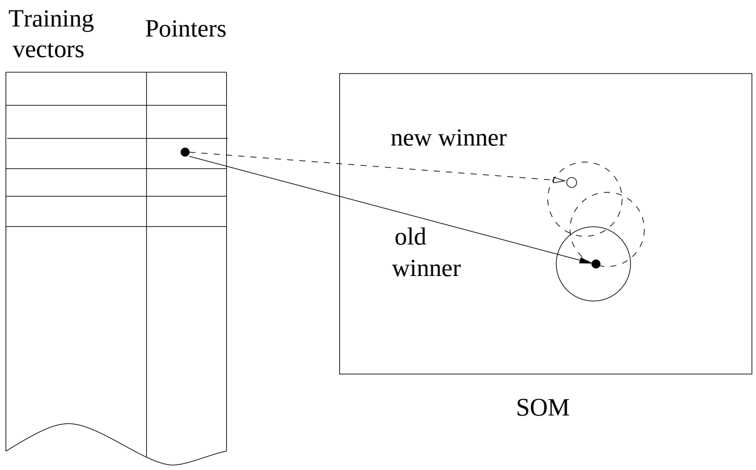

Figure 11: Finding the new winner in the vicinity of the old one, whereby the old winner is directly located by a pointer. The pointer is then updated

Assume further that the SOM is already smoothly ordered although not yet asymptotically stable. This is the situation, e.g., during the lengthy fine-tuning phase of the SOM, whereby the size of the neighborhood set is also constant and small. If, after inputting a particular input, updating of a number of map units is made before the same training input is used again some time later, it may be clear that the new winner is found at or in the vicinity of the old one. Therefore, in searching for the best match, it will suffice to locate first the map unit corresponding to the associated pointer, and then to perform a local search for the winner in the neighborhood around the located unit. This will be a significantly faster operation than an exhaustive winner search over the whole SOM. The search can first be made in the immediate surround of the said location, and only if the best match is found at its edge, searching is continued in the surround of the preliminary best match, until the winner is one of the middle units in the search domain. After the new winner location has been identified, the associated pointer in the input table is replaced by the pointer to the new winner location.

For instance, if the array topology of the SOM is hexagonal, the first search might be made in the 7-neighborhood of the winner. If the tentative winner is one of the edge units of this neighborhood, the search must be continued in the new 7-neighborhood centered around the last tentative winner (for the three map units that have not yet been checked), etc.

This principle can be used with both the usual incremental-learning SOM and its batch computing version.

A benchmark with two large SOMs relating to our recent practical experiments was made. The approximate codebook vector values were first computed by the CNAPS computer, whereafter they were fine-tuned by a general-purpose computer. During this fine-tuning phase, the radius of the neighborhood set in the hexagonal lattice decreased linearly from 3 to 1 units equivalent to the smallest lattice spacings, and the learning-rate factor at the same time decreased linearly from 0.02 to zero. There were 3645 training vectors for the first map, and 9907 training vectors for the second map, respectively. The results are reported in Table 4.

Input dimensionality	Map size	Speedup factor in winner search	Speedup factor in training
270	315	43	14
315	768	93	16

Table 4: Speedup due to shortcut winner search.

The theoretical maximum of speedup in winner search is: 45 for the first map, and 110 for the second map, respectively. The training involves the winner searches, codebook updating, and overhead times due to the operating system and the SOM software used. The latter figures may be improved by optimization of computing.

9.2 Estimating Initial Values for a Large SOM

Several suggestions for "growing SOMs" (cf., e.g. [3-5]) have been made. The detailed idea presented below has been optimized in order to make very large maps, and is believed to be new. The basic idea is to estimate good initial values for a map that has plenty of units, on the basis of asymptotic values of a map with a much smaller number of units.

As the general nature of the SOM process and its asymptotic states is now fairly well known, we can utilize some "expert knowledge" here. One fact is that the asymptotic distribution of codebook vectors is generally smooth, at least for a continuous,

smooth probability density function (pdf) of input, and therefore the lattice spacings can be smoothed, interpolated, and extrapolated locally.
As an introductory example consider, for instance, the one-dimensional SOM and assume tentatively a uniform probability density function (pdf) of the scalar input in the range $[a, b]$. Then we have the theoretical asymptotic codebook values for different numbers of map units that approximate the same pdf, as shown in Fig. 12.

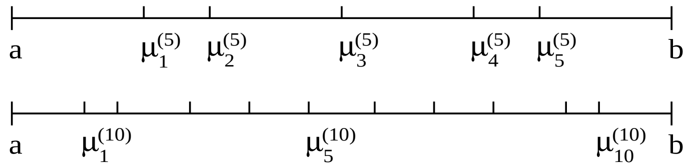

Figure 12: Asymptotic values for the μ_i for different lengths of the array, shown graphically

Assume now that we want to estimate the locations of the codebook values for an *arbitrary* pdf and for a 10-unit SOM on the basis of known codebook values of the 5-unit SOM. A linear *local* interpolation-extrapolation scheme can then be used. For instance, to interpolate $\mu_5^{(10)}$ on the basis of $\mu_2^{(5)}$ and $\mu_3^{(5)}$, we first need the *interpolation coefficient* λ_5, computed from the two ideal lattices with uniform pdf:

$$\mu_5^{(10)} = \lambda_5 \mu_2^{(5)} + (1 - \lambda_5)\mu_3^{(5)} , \tag{73}$$

from which λ_5 for $\mu_5^{(10)}$ can be solved. If then, for an arbitrary pdf, the *true* values of $\mu'^{(5)}_2$ and $\mu'^{(5)}_3$ have already been computed, the estimate of the true $\hat{\mu}_5'^{(10)}$ is

$$\hat{\mu}_5'^{(10)} = \lambda_5 \mu'^{(5)}_2 + (1 - \lambda_5)\mu'^{(5)}_3 . \tag{74}$$

Notice that a similar equation can also be used for the *extrapolation* of, say, $\mu_1^{(10)}$ on the basis of $\mu_1^{(5)}$ and $\mu_2^{(5)}$.
Application of local interpolation and extrapolation to *two-dimensional* SOM lattices (rectangular, hexagonal, or other) is straightforward, although the expressions become a little more complicated. Interpolation and extrapolation of a codebook vector in a two-dimensional lattice must be made on the basis of vectors defined at least in *three* lattice points. As the maps in practice may be very nonlinear, the best estimation results are usually obtained with three reference vectors.
Consider a pdf that is uniform over a two-dimensional rectangular area, approximated by two different overlapping "ideal" two-dimensional SOM lattices with the codebook vectors $\mathbf{m}_h^{(d)} \in \Re^2, \mathbf{m}_i^{(s)} \in \Re^2, \mathbf{m}_j^{(s)} \in \Re^2$, and $\mathbf{m}_k^{(s)} \in \Re^2$ its nodes, where the superscript d refers to a "dense" lattice, and s to a "sparse" lattice, respectively. If $\mathbf{m}_i^{(s)}, \mathbf{m}_j^{(s)}$, and $\mathbf{m}_k^{(s)}$ do not lie on the same straight line, then in the two-dimensional signal plane any $\mathbf{m}_h^{(d)}$ can be expressed as the linear combination

$$\mathbf{m}_h^{(d)} = \alpha_h \mathbf{m}_i^{(s)} + \beta_h \mathbf{m}_j^{(s)} + (1 - \alpha_h - \beta_h)\mathbf{m}_k^{(s)} , \tag{75}$$

where α_h and β_h are interpolation-extrapolation coefficients. This is a two-dimensional vector equation from which the two unknowns α_h and β_h can be solved.

Consider then a pdf in a space of arbitrary dimensionality and two SOM lattices with the same topology as in the ideal example. When the true pdf is arbitrary, we may not assume the lattices of true codebook vectors as planar. Nonetheless we can perform a *local linear estimation* of the true codebook vectors $\mathbf{m}'^{(d)}_h \in \Re^n$ of the "dense" lattice on the basis of the true codebook vectors $\mathbf{m}'^{(s)}_i$, $\mathbf{m}'^{(s)}_j$, and $\mathbf{m}'^{(s)}_k \in \Re^n$ of the "sparse" lattice.
In practice, in order that the linear estimate be most accurate, we may stipulate that the respective indices h, i, j, and k are such that in the ideal lattice $\mathbf{m}^{(s)}_i, \mathbf{m}^{(s)}_j$, and $\mathbf{m}^{(s)}_k$ are the three codebook vectors *closest* to $\mathbf{m}^{(d)}_h$ in the signal space (but not on the same line). With α_h and β_h solved from (75) for each node h separately we obtain the wanted interpolation-extrapolation formula as

$$\hat{m}'^{(d)}_h = \alpha_h m'^{(s)}_i + \beta_h m'^{(s)}_j + (1 - \alpha_h - \beta_h) m'^{(s)}_k \ . \tag{76}$$

Notice that the indices h, i, j, and k refer to *topologically identical* lattice points in (75) and (76). The interpolation-extrapolation coefficients for two-dimensional lattices depend on their topology and the neighborhood function used in the last phase of learning. For the "sparse" and the "dense" lattice, respectively, we have to compute first the ideal two-dimensional codebook vector values. As the closed solutions may be very difficult to obtain, the asymptotic codebook vector values may be solved by simulation. If the ratio of the horizontal vs. vertical dimensions of the lattice is $H : V$, we may draw two-dimensional input vectors at random from a uniform, rectangular pdf, the width of which in the horizontal direction is H and the vertical width of which is V.

References

[1] P. Koikkalainen. Progress with the tree-structured self-organizing map. In *Proc. ECAI 94, 11th European Conf. on Artificial Intelligence*, A. Cohn (ed.), John Wiley & Sons, pp. 211-215, 1994.

[2] P. Koikkalainen. Fast determenistic self-organizing maps. In *Proc. ICANN, Int. Conf. on Artificial Neural Networks*, Paris, France, vol. 2, pp. 63-68, 1995.

[3] J.S. Rodrigues and L.B. Almeida. Improving the learning speed in topological maps of patterns. In *Proc. INNC'90, Int. Neural Networks Conference.* Kluwer, Dordrecht, Netherlands, pp. 813-816, 1990.

[4] B. Fritzke. Let it grow - self-organizing feature maps with problem dependent cell structure. In *Artificial Neural Networks*, T. Kohonen, K. Mäkisara, O. Simula, J. Kangas (eds.). North-Holland, Amsterdam, Netherlands, pp. I-403-408, 1991.

[5] J. Blackmore and R. Miikkulainen. Incremental grid growing: Encoding high-dimensional structure into a two-dimensional feature map. In *Proc. ICNN'93, Int. Conf. on Neural Networks.* IEEE, Piscataway, NJ, pp. I-450-455, 1993.