Neural Processing Letters **5:** 153–159, 1997.

Interweaving Kohonen Maps of Different Dimensions to Handle Measure Zero Constraints on Topological Mappings

L. MANEVITZ
Department of Mathematics and Computer Science, University of Haifa, Haifa 31905, Israel
E-mail: manevitz@mathcs2.haifa.ac.il

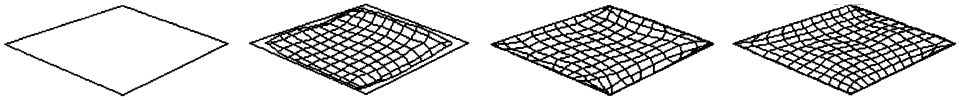
Key words: finite element method, Kohonen, neural networks, self-organizing, topological mapping

Abstract. The usual 'Kohonen' algorithm uses samples of points in a domain to develop a topological correspondence between a grid of 'neurons' and a continuous domain. 'Topological' means that near points are mapped to near points. However, for many applications there are additional constraints, which are given by sets of measure zero, which are not preserved by this method, because of insufficient sampling. In particular, boundary points do not typically map to boundary points because in general the likelihood of a sample point from a two-dimensional domain falling on the boundary is typically zero for continuous data, and extremely small for numerical data. A specific application, (assigning meshes for the finite element method), was recently solved by interweaving a two-dimensional Kohonen mapping on the entire grid with a one-dimensional Kohonen mapping on the boundary. While the precise method of interweaving was heuristic, the underlying rationale seems widely applicable. This general method is problem independent and suggests a direct generalization to higher dimensions as well.

1. Basic Problem

The Kohonen algorithm [2] is designed to produce a 'topological' map from a space of 'neurons' to another domain. This is achieved by taking *representatives* of the domain and applying a 'competitive' algorithm between the 'neurons' to determine a winner and then adjusting the weights of both the neuron and its neighbors. 'Representatives' can mean passing through all members of the domain or taking a sample. The neurons of course fit the distribution of the sample set, and so the success of the method typically depends on the quality of the sample set representing the full domain.

Unfortunately, there are occasionally features of a domain which are not represented well in a data set because the points determining the feature are of measure zero in the domain. One of the most outstanding examples of this is the property of being on the boundary. That is, it is *not* the case that running the Kohonen algorithm will typically send boundary points to boundary points of the domain. Looking at any of the standard 'textbook' examples, we see that the boundary is not actually reached by the neurons themselves. Of course, the topologically preserving



A. Domain of Application

B. Result of Standard Kohonen Mapping

C. Result of Sampled Boundary Algorithm

D. Result of Interwoven 1-D and 2-D Kohnen Map

Figure 1. Comparison of different algorithms.

properties of the mesh tend to force the boundary of the neural network to remain on the boundary of the *network*. Moreover, there is a distortion of the ultimate and penultimate positions of the neural space. (This phenomenon was, of course, observed by Kohonen e.g. [2], p. 134.) Figure 1B illustrates this phenomenon.

In many applications the constraint that boundary points of the network should lie on boundary points of the domain is an important constraint and the Kohonen algorithm as generally presented is thus not well suited for developing such maps. (One can imagine other such 'measure zero' constraints, e.g. the need to have points on a 'crack', etc.)

One such application, mesh generation for the finite element method, has such a constraint, and in its recent solution [4] it was necessary to arrange that: (1) the mesh should represent the domain (which need not necessarily be convex); (2) boundary points of the network should lie on the boundary of the domain; (3) emphasis placed as needed on certain areas of the domain; i.e. the density of representation is not necessarily uniform. (4) the 'quality' of the resultant mesh, in finite element terms, should be high. This means that certain rules regarding aspect ratios, continuous change in size of elements, etc. should be maintained.[1]

In this application, points (1), (3) and (4) were handled by essentially the usual Kohonen algorithm with appropriate problem dependent modifications.[2] For example, point (3) is handled by using an appropriate non-uniform density function to generate the sampling set. Point (4) also follows more or less automatically by the basic design of the Kohonen map.

Full details of methods and results appear in [4]; some sample results appear below. Here our interest is to describe briefly the ideas involved in point (2).

2. Boundaries

Basically, the above analysis indicates that one needs to sample the boundary points sufficiently often to guarantee that points will reach the boundary. In an earlier version, this was accomplished by simply artificially guaranteeing that roughly one out of every seven points are chosen from the boundary. This is in fact sufficient to guarantee that boundary points of the network fall on the boundary; unfortunately, this is a gross distortion of the desired density function and results therefore in unsatisfactory results for nodes near the boundary. Figure 1C shows the result of this mechanism.

To counter-act this affect, two mechanisms were added: (1) The running of a secondary Kohonen mapping between the two one-dimensional subspaces (i.e. between the boundary nodes of the neural network and the boundary of the domain). (2) An appropriate balancing between the two Kohonen mappings.

The first point works directly. The second was solved experimentally and heuristically.

The method used in this problem involves choosing

- appropriate schedules for the Kohonen adjustment parameter for each of the two Kohonen mappings;
- the appropriate interweaving between the two different Kohonen algorithms;
- deciding how to 'decouple' the two algorithms and at which state.

The rationale for the approach was to first use the known method of obtaining a boundary numerically by artificially running the two-dimensional Kohonen map with a distorted density function obtained by choosing one out of seven points on the boundary. This was run sufficiently long to place points reasonably close to the boundary.

Then the one-dimensional map between the boundary and the neurons on the boundary of the mesh was started; interweaving it with the on-going two dimensional map. This helps to place the boundary points in appropriate places. However, movement of the one-dimensional map and of the two-dimensional maps still affect all points. After a 'while', the two maps are decoupled; points on the boundary are no longer affected by movements in the interior of the domain. Essentially this means that the two-dimensional Kohonen map now works only on the interior of the neural network.

The details of the specific application were obtained after experimentation with different values and need not be valid for other problems; they are listed here to give an idea of the parameters involved. The details involve:

1) running a 'pure' Kohonen algorithm on the domain for around 1500 iterations. Boundaries are typically not reached during this stage;
2) between 1500 to 3000 iterations running a 'distorted' Kohonen algorithm by choosing one out of every 7 sample points to be on the boundary of the domain (actually the algorithm would alternate choosing 30 interior points and then 5 boundary points);
3) between 3000 to 5000 iterations running both the 1-D and the 2-D Kohonen maps; again with a ratio of one out of every 7 points chosen on the boundary;
4) over 5000 iterations running both the 1-D and the 2-D Kohonen maps. However, at this point, points on the boundary can no longer be moved into the interior. This effectively means that the 2-D Kohonen map is now running only on the original mesh minus its boundary;
5) the 1-D Kohonen parameter α' is adapted independently of the 2-D Kohonen parameter α. See below for the table of values per iteration actually used for these parameters.

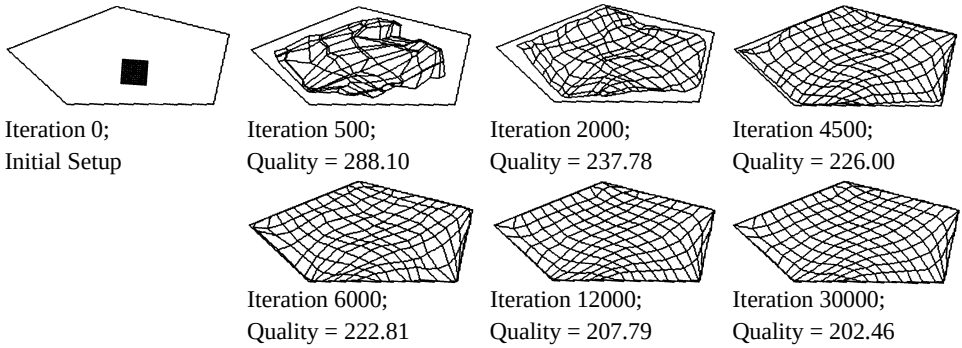


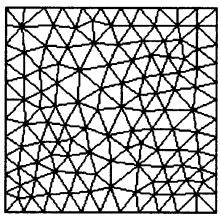
Figure 2. A network developing over time.

Iterations	Value of α	Iterations	Value of α
1–9	0.8	3000–3999	.45
10–999	0.3	4000–4999	.35
1000–1999	0.2	5000–6000	.25
2000–3000	0.1	6001–10000	0.10
3001–7000	0.05	10001–11000	.11
7001–8000	0.08	11001–13000	.03
8001–10000	0.01	over 10000	.007
over 10000	0.001		

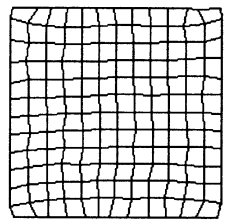
Figure 1D shows the same example run under this interwoven algorithm. Figure 2 shows an example of a mesh developing over time for a uniform density function. Note that at 2000 iterations, points have not yet reached the boundary; at 6000 iterations the boundary is essentially covered but the density is distorted; by 30000 iterations it begins to look fairly regular.

[4] reports on tests judging the efficacy of this method; it was compared with a fully developed well-known system PLTMG [1] designed to generate meshes. Since PLTMG produces triangular meshes, the comparison was between the quality of solutions produced by the two systems of a series of boundary value problems, over different domains both convex and nonconvex; and some with non-uniform density. In general this algorithm was somewhat superior to PLTMG. Below a few sample results are presented; one with a nonconvex domain and one with a nonuniform density. In each case the boundary value problem has analytic solutions so that one can measure both the average error per node and the error per value of the solution function. On these measures our algorithm (denoted NN) was roughly an order of magnitude superior on most examples. (u is the analytic solution; u_h

is the numerically computed solution; $Error/Node = \frac{\sum_{nodes}|u(node)-u_h(node)|}{\#(nodes)}$; $Error/Value = \frac{\sum_{nodes}|u(node)-u_h(node)|}{\sum_{nodes}|u(node)|}$.)



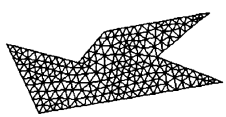
PLTMG 139 nodes



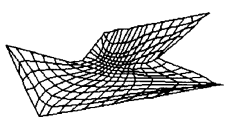
NN 169 nodes; 55000 Iterations

Rectangular domain PLTMG (139 nodes, 232 elements) NN (169 nodes, 144 elements)

		Error/Node		Error/Value	
$u(x,y)$	$f(x,y)$	PLTMG	NN	PLTMG	NN
x^3+y^3	$-6x-6y$	6.427036E-03	7.635503E-04	2.082413E-04	2.522708E-05
$sinx+siny$	$sinx+siny$	3.085165E-04	2.279894E-04	2.68856E-04	1.954924E-04



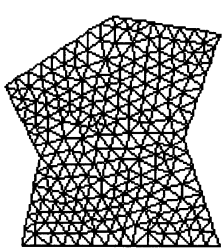
PLTMG 234 nodes



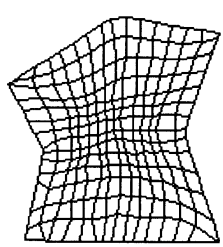
NN 256 nodes; quality = 381.146362; 'hot-spot' (2,2) near center; 30000 iterations

7-Sided nonconvex domain PLTMG (234 nodes, 385 elements) NN (256 nodes, 225 elements)

		Error/Node		Error/Value	
$u(x,y)$	$f(x,y)$	PLTMG	NN	PLTMG	NN
$e^{-(x-2)^2}e^{-(y-2)^2}$	$-u_{xx}-u_{yy}$	2.864687E-02	1.811124E-03	9.151940E-02	1.159691E-03



PLTMG 249 nodes



NN 225 nodes; quality = 278.713715; 'hot-spot' (2,2) near center; 46650 iterations

7-Sided non-convex domain PLTMG (249 nodes, 437 elements) NN (225 nodes, 196 elements)					
		Error/Node		Error/Value	
$u(x,y)$	$f(x,y)$	PLTMG	NN	PLTMG	NN
$e^{-(x-2)^2}e^{-(y-2)^2}$	$-u_{xx}-u_{yy}$	2.412143E-02	7.530449E-03	4.515054E-02	9.097765E-03

3. Summary and Future Suggestions

A problem using Kohonen maps on a two-dimensional domain that *requires* satisfying a one-dimensional constraint was recently solved by interweaving developing different dimensional Kohonen maps simultaneously.

It seems reasonable that this method can also work for other problems where there are rare subdomains which should be mapped by subdomains of Kohonen neural networks.

For example, if one wanted to take apply the Kohonen mapping to produce good quality meshes in three dimensions (or more) the same ideas can be used.

The basic algorithm would then be modified to

1. Place a 3-dimensional simplex, e.g. boxes, in the center of the domain.
2. Run a 3-dimensional version of the Kohonen algorithm until the simplex reaches or comes 'close' to the 2-dimensional boundaries.
3. Then interweave 2-dimensional versions of the Kohonen algorithm with the 3 dimensional one until the 1-dimensional boundaries are reached. This requires knowing when to decouple the two algorithms, the relative frequency of work on each algorithm and the appropriate Kohonen parameters.
4. Then interweave 1-dimensional versions of the Kohonen algorithm on each of these 1-dimensional boundaries, together with the 2-Dimensional and the 3-dimensional algorithms. Again one has to know when to decouple the two algorithms, etc.

The precise details of interweaving however, even in this direct generalization of the work presented in [4], would seem to require substantial experimentation.

Notes

1. More specifically, for finite element meshes there are known heuristic requirements for the quality of the mesh. Formalizing these mathematically for a mesh consisting of quadrilaterals, one has the following formula (where for a given element a^e refers to the largest side of the quadrilateral, b^e refers to the smallest. $E_1^e = 1 - b^e/a^e$, giving a measure of the aspect ratio, $E_2^e = max_{i=1}^4 |1 - \frac{2}{\pi} angle_i|$ measuring how close all the quadrilateral angles are to $90^o s$, $E_3^e = max_{neighbors} \left|1 - min_{n=1}^{N^e} \{a^e/a_n^e, a_n^e/a^e\}\right|$, measuring how similar an element is to its neighboring ones; w_i are positive weights for the different measures)

$$Quality(Mesh) = \sum_{elements} w_1 E_1^e + w_2 E_2^e + w_3 E_3^e.$$

   (In [4] lacking any further information all the w_is were taken to have value 1.)
2. A speed-up suggestion of Tabakman and Exman [3] was also implemented.

References

1. R.E. Bank, PLTMG: A Software Package for Solving Elliptic Partial Differential Equations, SIAM publications: Philadelphia, 1994.
2. T. Kohonen, Self-Organization and Associative Memory, second edition, Springer-Verlag: Berlin, 1988.
3. T. Tabakman and I. Exman, "Towards real-time self-organizing maps with parallel and noisy inputs", Proceedings of the 10th Israeli Symposium on Artificial Intelligence, Computer Vision and Neural Networks, pp. 155–164, Ramat Gan, Israel, 1993.
4. L. Manevitz, Malik Yousef and D. Givoli, "Finite element mesh generation using self-organizing neural networks", Microcomputers in Civil Engineering, Vol. 12, pp. 233–251, 1997.