The Parameterless Self-Organizing Map Algorithm

Erik Berglund and Joaquin Sitte

***Abstract*—The parameterless self-organizing map (PLSOM) is a new neural network algorithm based on the self-organizing map (SOM). It eliminates the need for a learning rate and annealing schemes for learning rate and neighborhood size. We discuss the relative performance of the PLSOM and the SOM and demonstrate some tasks in which the SOM fails but the PLSOM performs satisfactory. Finally we discuss some example applications of the PLSOM and present a proof of ordering under certain limited conditions.**

***Index Terms*—Self-organizing feature maps.**

I. Introduction

THE self-organizing map (SOM) [1], [2] is an algorithm for mapping from one (usually high-dimensional) space to another (usually low-dimensional) space. The SOM learns the correct mapping independent of operator supervision or reward functions that are seen in many other neural network algorithms, e.g., backpropagation perceptron networks. Unfortunately this unsupervised learning is dependent on two annealing schemes, one for the learning rate and one for the neighborhood size. There is no firm theoretical basis for determining the correct type and parameters for these annealing schemes, so they must often be determined empirically. The generative topographic mapping (GTM) [3]–[5] is one attempt at addressing this. Furthermore, since these annealing schemes are time-dependent, they prevent the SOM from assimilating new information once the training is complete. While this is sometimes a desirable trait, it is not in tune with what we know of the adaptive capabilities of the organic sensomotor maps which inspired the SOM [6]. There have been several attempts at providing a better scaling method for learning rate and/or neighborhood size as well as taking some of the guesswork out of the parameter estimation.

A. Previous Works

One such attempt was done by Göppert and Rosenstiel [7], where the SOM is used to approximate a function, and the approximation is used as a neighborhood size decay parameter on a per-node basis. Unfortunately this is not applicable to cases other than function approximation and it requires knowledge of the desired approximation values of the function, thus losing the major advantage of the SOM; unsupervised learning. A more closely related approach would be the plastic SOM (PSOM) [8], where the Euclidean distance from the input to the weight vector of the winning node is used to determine whether to add new nodes to the map. This is similar to the growing neural gas (GNG) algorithms [9], [10] but maintains plasticity. Another approach is the self-organizing with adaptive neighborhood neural network (SOAN) [11] which calculates the neighborhood size in the input space instead of the output space like the SOM variants. The SOAN tracks the accumulated error of each node, and scales the neighborhood function accordingly between a minimum and a maximum value, and like the GNG algorithms it can increase or decrease the number of nodes. This still leaves several parameters to be determined empirically by the user. The time-adaptive SOM (TASOM) [12], [13] addresses the inability of the SOM to maintain plasticity by keeping track of dynamic learning rates and neighborhood sizes for each individual node. The neighborhood size is dependent on the average distance between the weight vector of the winning node c and its neighbors, while the learning rate is dependent only on the distance between the weight of a given node i and the input, similar to the standard SOM. The user is still required to select several training parameters without firm theoretical basis. The auto-SOM [14], [15] uses Kalman filters to guide the weight vectors toward the center of their respective Voronoi cells in input space. This automates computation of learning rates and neighborhood sizes, but the user is still required to set the initial parameters of the Kalman filters. Unfortunately it is more computationally expensive than the SOM, and this problem increases with input size and number of inputs in the training set. The auto-SOM also needs to keep track of all previous inputs, which makes continuous learning difficult and increases computational load, or compute the Voronoi set for each iteration, which would increase computational load and is only feasible if the input probability density distribution is known. Other recent developments in self-organization include the self-organizing learning array [16] and noisy self-organizing neural Networks [17].

B. Overview

For these reasons, we introduce the parameterless SOM (PLSOM). The fundamental difference between the PLSOM [18] and the SOM is that while the SOM depends on the learning rate and neighborhood size to decrease over time, e.g., as a function of the number of iterations of the learning algorithm, the PLSOM calculates these values based on the local quadratic fitting error of the map to the input space. This allows the map to make large adjustments in response to unfamiliar inputs, i.e., inputs that are not well mapped, while not making large changes in response to inputs it is already well adjusted to. The fitting error is based on the normalized distance from the input to the weight vector of the winning node in input space. This value (referred to as ϵ, the lowercase Greek letter epsilon, throughout this paper) is computed in any case,,

Manuscript received October 25, 2004; revised May 15, 2005.

E. Berglund is with the Division of Complex and Intelligent Systems, Information Technology and Electrical Engineering, University of Queensland, St. Lucia QLD 4072, Australia (e-mail: berglund@itee.uq.edu.au).

J. Sitte is with the Smart Devices Laboratory, Queensland University of Technology, Brisbane QLD 4001, Australia (e-mail: j.sitte@qut.edu.au).

Digital Object Identifier 10.1109/TNN.2006.871720

hence, this mechanism can be implemented without inducing noteworthy increases in the computational load of the map or hindering parallelised implementations [19]. In Section II, we give details of the PLSOM algorithm, in Section III we evaluate its performance relative to the SOM. In Section IV, we explain the observed behavior of the PLSOM. In Section V, we give examples of applications, and in Section VI, there is a brief discussion of some aspects of the PLSOM relative to nonlinear mapping problems. Section VII is the conclusion. For mathematical proofs, see Appendix I.

II. Algorithm

As an introduction and to give a background for the PLSOM we will here give a brief description of the SOM algorithm before we move on to the PLSOM itself.

A. Ordinary SOM Algorithm

The SOM variant we will examine is the Gaussian-neighborhood, Euclidean distance, rectangular topology SOM, given by (2)–(6). The algorithm is, in brief, as follows: An input $x(t)$ is presented to the network at time (or timestep, iteration) t. The 'winning node' $c(t)$, i.e., the node with the weight vector that most closely matches the input at time t, is selected using (1)

$$c(t) = \arg\min_i \left(\|x(t) - w_i(t)\|_2\right) \tag{1}$$

where $w_i(t)$ is the weight vector of node i at time t. $\|.\|_2$ denotes the L^2-norm or n-dimensional Euclidian distance. (The SOM can use other distance measures, e.g., Manhattan distance.) The weights of all nodes are then updated using (2)–(4)

$$w_i(t+1) = w_i(t) + \Delta w_i(t) \tag{2}$$

$$\Delta w_i(t) = \alpha(t) h_{c,i}(t)\left[x(t) - w_i(t)\right] \tag{3}$$

$$h_{c,i}(t) = e^{\frac{-d(i,c)^2}{\beta(t)^2}} \tag{4}$$

where $h_{c,i}(t)$ is referred to as the neighborhood function, and is a scaling function centred on the winning node c decreasing in all directions from it. $d(i,c)$ is the Euclidean distance from node i to the winning node c in the node grid. As is the case with the input/weight distance, the node distance can be calculated using some other distance measure than the Euclidean distance, e.g., the Manhattan distance or the link distance, and the grid need not be rectangular. $\alpha(t)$ is the learning rate at time t, $\beta(t)$ is the neighborhood size at time t.

Lastly, the learning rate (α) and neighborhood size (β) are decreased in accordance with the annealing scheme. One possible annealing scheme is given by (5) and (6) for the decrease of the learning rate and the neighborhood size, respectively. The important point is that the annealing scheme relies on the time step number t and not the actual fitness of the network

$$\alpha(t+1) = \alpha(t)\delta_\alpha, \qquad 0 < \delta_\alpha < 1 \tag{5}$$

$$\beta(t+1) = \beta(t)\delta_\beta, \qquad 0 < \delta_\beta < 1. \tag{6}$$

Here, δ_β and δ_α are scaling constants determined beforehand.

These steps are repeated until some preset condition is met, usually after a given number of iterations or when some measurement of error reaches a certain level. The density of the nodes in input space are proportional to the density of input samples, however this may lead to undesired results, see Fig. 9. Several variations of the algorithm outlined here exists, e.g., the Matlab implementation of the SOM uses a two-phased learning algorithm (an ordering phase and a tuning phase) and a step-based neighborhood function.

B. PLSOM Algorithm

The fundamental idea of the PLSOM is that amplitude and extent of weight updates are not dependent on the iteration number, but on how well the PLSOM fits the input data. To determine how good the fit is, we calculate a scaling variable which is then used to scale the weight update. The scaling variable, ϵ, is defined in (7) and (8)

$$\epsilon(t) = \frac{\|x(t) - w_c(t)\|_2}{r(t)} \tag{7}$$

$$r(t) = \max\left(\|x(t) - w_c(t)\|_2, r(t-1)\right)$$

$$r(0) = \|x(0) - w_c(0)\|_2 . \tag{8}$$

$\epsilon(t)$ is best understood as the normalized Euclidean distance from the input vector at time t to the closest weight vector. If this variable is large, the network fits the input data poorly, and needs a large readjustment. Conversely, if ϵ is small, the fit is likely to already be satisfactory for that input and no large update is necessary.

The algorithm for the PLSOM uses a neighborhood size determined by ϵ, thus replacing the equation governing the annealing of the neighborhood with $\beta(t) = constant\ \forall t$. β is scaled by $\epsilon(t)$ in the manner of (9), giving $\Theta(\epsilon(t))$, the scaling variable for the neighborhood function (12)

$$\Theta\left(\epsilon(t)\right) = \beta\epsilon(t) \quad \Theta\left(\epsilon(t)\right) \geq \theta_{\min}. \tag{9}$$

Equation (9) is not the only option for calculating Θ; another example is

$$\Theta\left(\epsilon(t)\right) = (\beta - \theta_{\min})\epsilon(t) + \theta_{\min}. \tag{10}$$

A third alternative is (11), which is used in generating Figs. 23(a)–23(d)

$$\Theta\left(\epsilon(t)\right) = (\beta - \theta_{\min}) ln\left(1 + \epsilon(t)(e-1)\right) + \theta_{\min} \tag{11}$$

where $ln()$ is the natural logarithm, e is the Euler number, and $\theta_{\min}$ is some constant, usually 0 for (11) or 1 for (9), (10). Equation (12) is the neighborhood function

$$h_{c,i}(t) = e^{\frac{-d(i,c)^2}{\Theta(\epsilon(t))^2}}. \tag{12}$$

As before, $d(i,c)$ is a distance measure along the grid, i.e., in output space, from the winning node c to i which is the node we are currently updating. This gives a value that decreases the further we get from c, and the rate of decrease is determined by ϵ, as can be seen in Fig. 1. The weight update functions are

$$w_i(t+1) = w_i(t) + \Delta w_i(t) \tag{13}$$

$$\Delta w_i(t) = \epsilon(t) h_{c,i}(t)\left[x(t) - w_i(t)\right]. \tag{14}$$

As we can see from (14) the learning rate $\alpha(t)$ is now completely eliminated, replaced by $\epsilon(t)$. Thus, the size of the update is not dependent on the iteration number. The only variable affecting the weight update which is carried over between iterations is the scaling variable $r(t)$. Practical experiments indicate

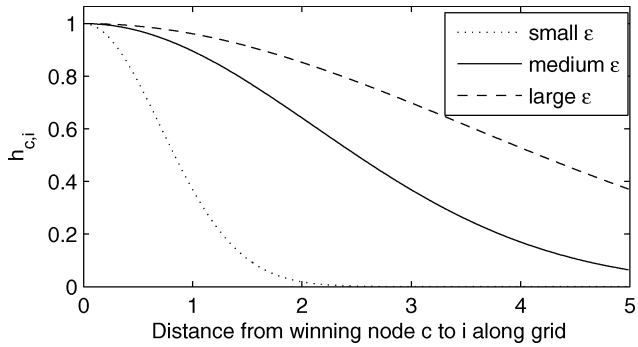


Fig. 1. Plot showing the effect of different ϵ values on the neighborhood function.

that r reaches it maximum value after the first few iterations, and does not change thereafter.

III. Performance

The PLSOM completely eliminates the selection of the learning rate, the annealing rate and annealing scheme of the learning rate and the neighborhood size, which have been an inconvenience in applying SOMs. It also markedly decreases the number of iterations required to get a stable and ordered map. The PLSOM also covers a greater area of the input space, leaving a smaller gap along the edges.

1) Comparison to the SOM Variants: We trained the Matlab SOM variant, the SOM and the PLSOM with identical input data, for the same number of iterations. The input data was pseudo-random, two-dimensional (2-D), and in the [0,1] range. This was chosen because a good pseudo-random number generator was readily available, eliminating the need to store the training data. Since the training data is uniformly distributed in the input space the perfect distribution of weight vectors would be an evenly spaced grid, with a narrow margin along the edges of the input space. That way, each weight vector would map an evenly sized area of the input space.

In comparing the two SOM implementations we used three separate quality measures, which are all based on the shape and size of the cells. A cell is the area in the input space spanned by the weight vectors of four neighboring nodes.

Unused space:

We summarized the area covered by all the cells, and subtracted this from the total area of the input space. The resulting graph clearly shows how the PLSOM spans a large part of the input space after only a small number of iterations and maintains the lead throughout the simulation (Fig. 2). Please note that this quality measure will be misleading in situations where cells are overlapping, but this will typically only occur in the first few thousand iterations.

Average skew:

For each cell we calculate the length of the two diagonals in a cell and divide the bigger by the smaller and subtract one, thus getting a number from 0 to infinity, where 0 represents a perfectly square cell. Again, we see that the PLSOM outperforms the SOM in the early stages of simulation but after ca. 24 000 iterations the SOM surpass the PLSOM. After 100 000 iterations the difference is still small, however (see Fig. 3).

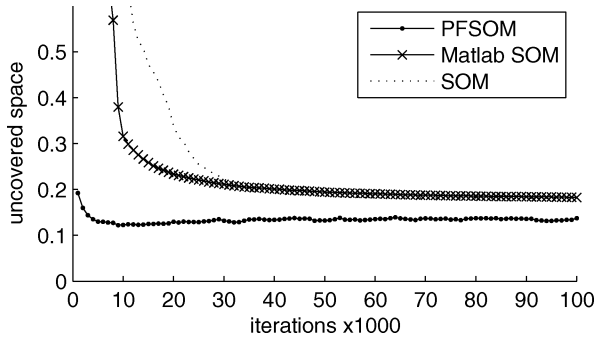


Fig. 2. Graph of the decrease of uncovered space as training progresses for the PLSOM, the SOM, and the Matlab SOM implementation. Note the quick expansion of the PLSOM and that it consistently covers a larger area than the SOM variants.

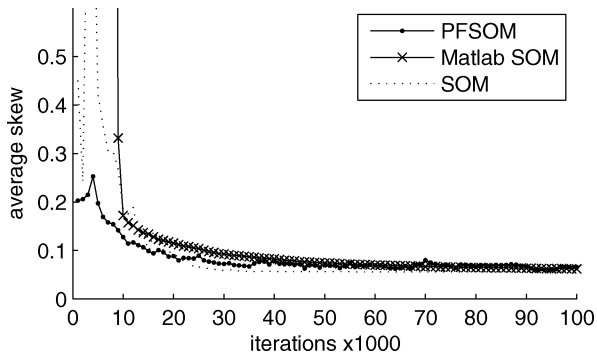


Fig. 3. Graph of the average skew for the PLSOM, the SOM, and the Matlab SOM implementation. For the first 24 000 iterations, the PLSOM is more ordered, before the SOM variants narrowly overtake it.

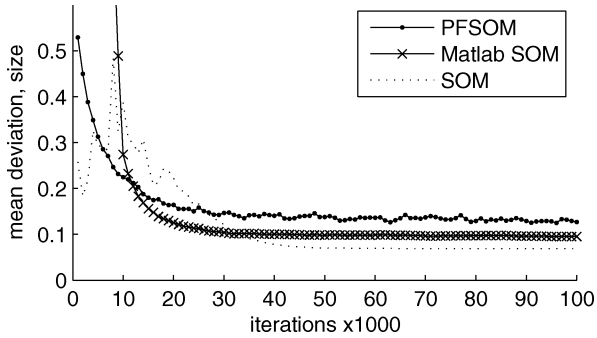


Fig. 4. Graph of the absolute mean deviation of cell size for the PLSOM, the SOM, and the Matlab SOM. The PLSOM is more regular up until ca. iteration 10 000.

Deviation of cell size:

We calculate the absolute mean deviation of the cell size and divide it by the average cell size to get an idea of how much the cells differ in relative size. Here the SOM is superior to the PLSOM after ca. 10 000 iterations, mainly because of the flattened edge cells of the PLSOM (see Fig. 4). If we ignore the cells along the edge, the picture is quite different: the PLSOM outperforms the SOM with a narrow margin (see Fig. 5).

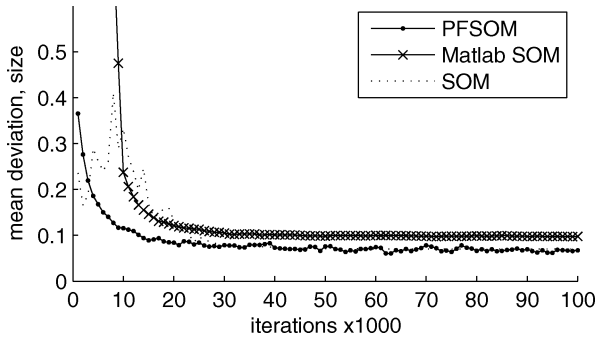


Fig. 5. Graph of the absolute mean deviation of cell size for the PLSOM, the SOM and the Matlab SOM, excluding the edge cells. Compare to Fig. 4. The PLSOM outperforms the Matlab SOM in both adaptation time and accuracy, and the SOM needs until ca. iteration 30 000 to reach the same level of ordering.

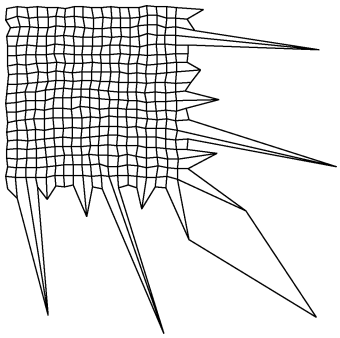


Fig. 6. SOM first trained with inputs ranging from 0 to 0.5 for 50 000 iterations shown after 20 000 further training iterations with inputs ranging from 0 to 1.0.

2) Plasticity Preservation: The illustrations in this section show the positions of the weight vectors, connected with lines, in the input space. When a SOM has been trained, it will not adapt well to new data outside the range of the training data, even if a small residual learning rate is left. This is illustrated by Fig. 6, where a SOM has been presented with pseudo-random, uniformly distributed 2-D data vectors in the [0, 0.5] range for 50 000 iterations. Thereafter the SOM was presented with 20 000 pseudo-random, uniformly distributed, 2-D data vectors in the [0, 1] range, after which the SOM has adapted very little to the new data. In addition the adaptation is uneven, creating huge differences in cell size and distorting the space spanned by the weight vectors. If we subject a PLSOM to the same changes in input range, the difference is quite dramatic; it adapts correctly to the new input range almost immediately, as seen in Fig. 7.

3) Memory: In the opposite case, *viz*. the SOM is presented with a sequence of inputs that are all restricted to a small area of the training input space, it would be preferable if the SOM maintains its original weight vector space, in order to not "forget" already learned data. Fig. 8 demonstrates what happens to a PLSOM if it is trained with pseudo-random, uniformly distributed 2-D data in the [0, 1] range for 50 000 iterations and then presented with inputs confined to the [0, 0.5] range for 20 000 iterations. This leads to an increase of the density of weight vectors in the new input space, yet maintains coverage of the entire initial input space, resulting in distortions along the edge of the new input space. Both these effects are most pronounced in the PLSOM.

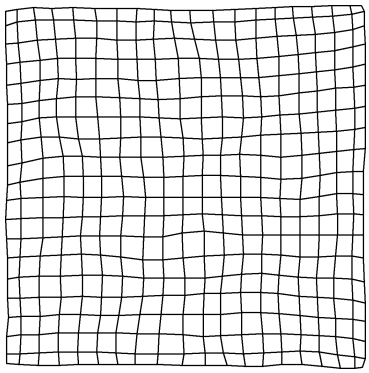


Fig. 7. PLSOM first trained with inputs ranging from 0 to 0.5 for 50 000 iterations shown after 20 000 further training iterations with inputs ranging from 0 to 1.0. Note the difference between this and Fig. 6.

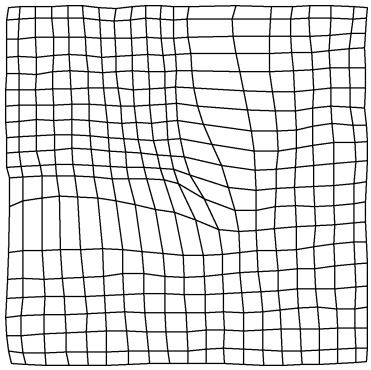


Fig. 8. PLSOM first trained with inputs ranging from 0 to 1 for 50 000 iterations shown after 20 000 further training iterations with inputs ranging from 0 to 0.5. Note that while the weights have a higher density in the new input space, the same area as before is still covered, i.e., none of the old input space has been left uncovered.

A. Drawbacks

The PLSOM is measurably less ordered than a properly tuned SOM and the edge shrinking is also more marked in the PLSOM. The PLSOM does not converge in the same manner as the SOM (there is always a small amount of movement), although this can be circumvented by not performing new weight updates after a satisfactory fit has been established.

IV. Analysis

This section highlights a special case where the SOM fails but the PLSOM succeeds, and explores the causes of this.

A. Experiments

We have applied the PLSOM and two variants of the SOM to the same problem; mapping a nonuniformly distributed input space. As input space we used a normal distributed pseudo-random function with a mean of 0.5 and standard deviation of 0.2. Values below 0 or above 1 were discarded. The same random seed was used for all experiments and for initializing weights. A SOM variant that uses the same neighborhood function as the PLSOM and an exponential annealing scheme for learning rate and neighborhood size, here nominated "plain SOM," was used for comparison. As can be seen from Fig. 9

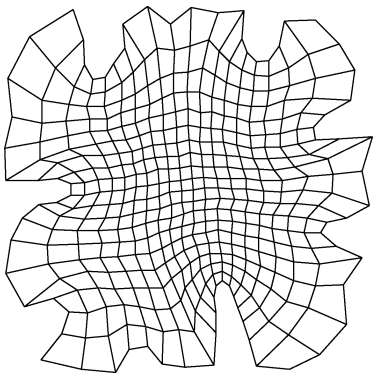


Fig. 9. Ordinary SOM after 100 000 iterations of normally distributed input with mean 0.5, standard deviation 0.2, clipped to the [0,1] interval. Note that two nodes which are close in input space may not be close on the map.

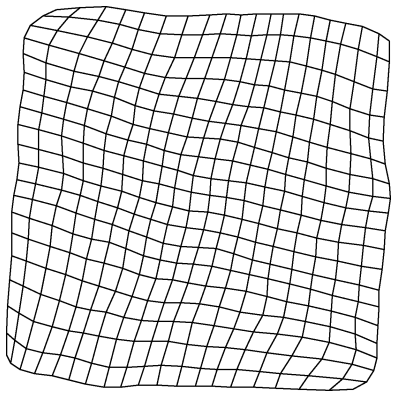


Fig. 10. PLSOM after 100 000 iterations of normally distributed input with mean 0.5, standard deviation 0.2, clipped to the [0,1] interval. While the correspondence between weight vector density and input density is weaker than for the SOM, the topology is preserved. Compare to Fig. 9. See also Fig. 15.

the SOM is severely twisted when we try a 20×20 node rectangular grid. The size of the ordinary SOM algorithm must be reduced to 7×7 before all traces of this twisting are removed. Altering the annealing time does not solve the problem. The PLSOM on the other hand performs well with the initial size of 20×20 nodes, filling the input space to over 77% (see Fig. 10).

B. Explanation

This phenomenon can be explained by looking at the likelihood of a given input in relation to the size of the weight update this input will result in, i.e., the expected update given the input distribution. The likelihood of an input occurring is governed by the Gaussian probability density function. The likelihood p of an input occurring in the interval $\langle z_1, z_2 \rangle$, where $z_1 \leq z_2$, is approximated using the error function erf

$$p(z_1, z_2) = \frac{1}{2}\left(\operatorname{erf}\left(\frac{z_2 s}{\sqrt{2}}\right) - \operatorname{erf}\left(\frac{z_1 s}{\sqrt{2}}\right)\right) \tag{15}$$

where s is a scaling constant to account for the standard deviation. An analysis of the expected update of a given node is given by

$$\xi(\Delta w, x) = \Delta w(x)\rho(x) \tag{16}$$

where $\xi(\Delta w, x)$ is the expected displacement of weight vector w given x as input, $\Delta w(x)$ is the displacement of w given x, and $\rho(x)$ is the probability density of this input.

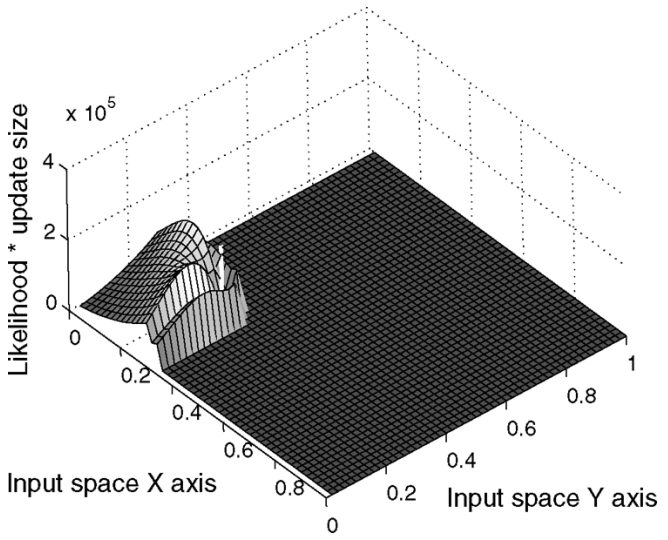


Fig. 11. Update size x likelihood for a corner node v of a 20×20 node ordinary SOM algorithm. The position of v in the input space is marked by a vertical white line. The position of v in the map is (1,1).

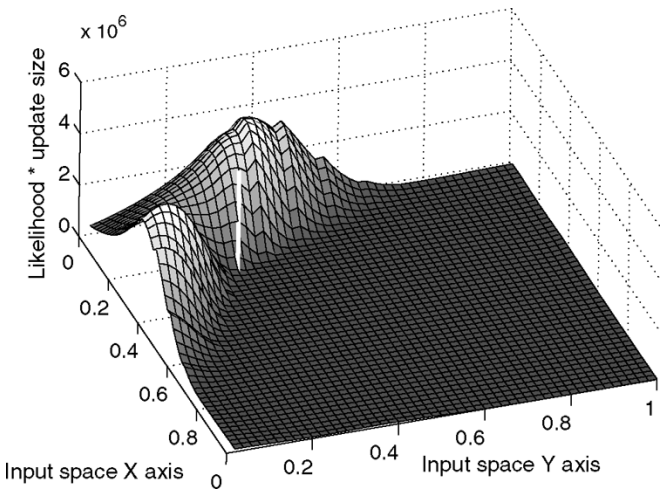


Fig. 12. Update size x likelihood for a corner node v of a 20×20 node PLSOM. The position of v in the input space is marked by a vertical white line. The position of v in the map is (1,1).

By discretising this over a 2-D $n \times n$ grid, N we can plot an approximation of the expected displacement for each square of N, as seen in Figs. 11 and 12.

When comparing the expected update of a Matlab SOM algorithm and a PLSOM we see that the PLSOM edge nodes receives a far larger amount of its update from outside the area covered by the map than its Matlab counterpart, thus making sure that the expansion outwards is even and less jerky.

To get a clearer picture, we need to integrate the expected displacement over the entire input space Ω which contains all possible inputs x, giving (17)

$$\xi(\Delta w) = \frac{1}{\Omega}\int_{\Omega} \Delta w(x)\rho(x)dx \tag{17}$$

Discretising the integrated expected displacement gives a vector for each node in the map, indicating how much and in which direction it is likely to be updated given the input distribution, as shown in Figs. 13 and 14.

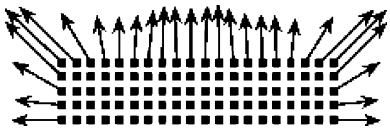


Fig. 13. Expected displacement vectors for the edge nodes along one edge of an ordinary SOM. Note that the vectors are changing direction abruptly from node to node, causing the warping.

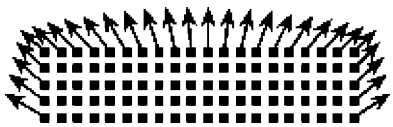


Fig. 14. Expected displacement vectors for the edge nodes along one edge of a PLSOM.

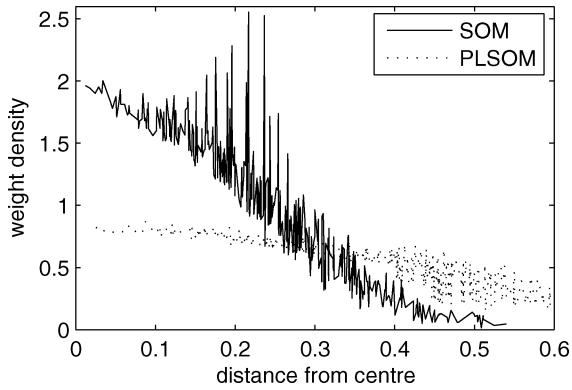


Fig. 15. Weight density versus distance from center for the SOM and the PLSOM. The 2-D input was normal distributed with a 0 mean and 0.2 standard deviation. Observe that while the PLSOM has less correlation between input density and weight density, it has far less variance and covers a larger area. See also Figs. 9 and 10.

As we can see from Fig. 13, this vector is greater for the corner node than for the side node in the SOM algorithm, while the opposite is true for the PLSOM, as seen in Fig. 14. This leads the corner nodes in the SOM algorithm to expand outwards faster than the side nodes, thus creating the warping. In the PLSOM, the side nodes expand outward faster, creating an initial 'rounded' distribution of the weights, but subsequent inputs pull the corners out. Also note that the edge nodes of the PLSOM is only marginally pulled inwards by inputs inside the weight grid, since the amount of update depends on the distance from the input to the closest node, not only on the distance from the node in question—this contributes to the quicker, more even expansion.

Finally, the weight update functions of the different algorithms give us the last piece of the explanation. Consider a map that receives an input far outside the area it is currently mapping, after already being partly through its annealing and, therefore, partially ordered.

When this happens to an ordinary SOM the update will be large for the winning node, but because the size of the neighborhood function is so small the neighbors of the winning node will receive only a very small update. If the same situation occurs to a PLSOM, the neighborhood size will scale (since it is dependent on the distance from the input to the winning node) to include a larger update for the neighbors of the winning node, thus distributing the update along a larger number of the edge nodes.

It should be pointed out that mapping a large portion of the input space while preserving such a skewed distribution is not possible—the difference between the length along the edges and the length through the center is too great to preserve neighborhood relations. When faced with this type of high-variance input distribution, one is faced with the choice of which property to sacrifice; neighborhood consistency or density equivalence. The SOM tries to do both, and fails. GNG, PSOM, and similar algorithms do both at the cost of ending up with arbitrary network connections. The PLSOM is unique in preserving neighborhood relations for a pre-defined network. This comes at a cost of poorer correspondence between input density and weight density, as can be seen from Fig. 15.

V. Applications

The PLSOM has been applied to three familiar problems by the authors. These applications will only be explored briefly here, in the interest of not distracting from the main subject of this article.

A. Sound Source Localization Through Active Audition

This application deals with processing a stereo sound signal, presenting it to a PLSOM or SOM to determine the direction of the sound source and orienting the microphones toward the sound source. This application illustrates that the PLSOM can deal with cases where the number of input dimensions (512) is far higher than the number of output dimensions (2).

1) Earlier Works: The physics of binaural audition is discussed in [20]–[22] and reinforcement learning in [23]. Early works include [24] and [25], but it is important do distinguish between passively determining the direction of sound sources and active audition. Active audition aims to using the movement of the listening platform to pinpoint the location, in the same way biological systems do. Active audition is, therefore, an intrinsically robotic problem. Other active audition works can be grouped into subcategories: First we have applications that rely on more than two microphones to calculate the source of a sound, e.g., [26]. While these are certainly effective, we observe that in nature two sound receivers are sufficient. Since microphones consume power and are a possible point of failure, we see definite advantages to being as frugal as nature in this respect. Secondly we have methods relying on synergy of visual and auditory cues for direction detection, most notably by members of the SIG Humanoid group [27]–[32]. Some of these also include neural networks and even SOMs, such as [33] and [34]. However, it is known that even humans that are born blind can accurately determine the direction of sounds [35], so interaction between vision and hearing cannot be a crucial component of learning direction detection. Our implementation is unique in that it does not rely on visual cues, specialised hardware nor any predefined acoustic model. It learns both the direction detection and the correct motor action through unsupervised learning and interaction with its environment. It also incorporates a larger

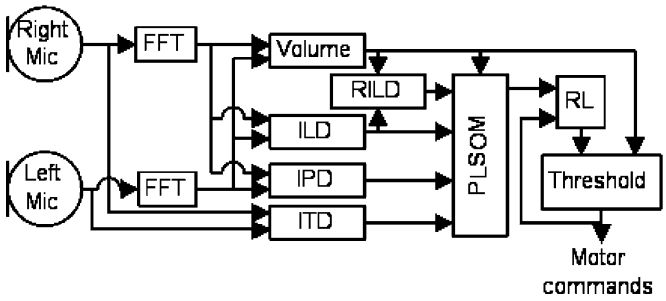


Fig. 16. Active audition system layout.

number of measures than other methods, which is made possible through the SOM/PLSOM ability to find patterns in high-dimensional data.

2) System Description: The aim is to let the process be completely self-calibrating—all that is needed is to provide a set of sound sources, and the algorithm will figure out on its own where the sound is coming from and how to orient toward it. This is done using a pipelined approach.

1) Digital stereo samples are streamed from a pair of microphones.
2) For each sampling window, which is 512 samples long, we compute the fast Fourier transform (FFT) of the signal. This is averaged to 64 subbands.
3) For each sampling window we compute interaural time difference (ITD), interaural level difference (ILD), interaural phase difference (IPD) and relative interaural level difference (RILD). ILD, IPD, and RILD are based on the FFT, so that we have one value for each of the 64 subbands.
4) The resulting 256-element vector is presented to a PLSOM.
5) The position of the winning node is used as index into a weight matrix which selects the appropriate motor action.
6) If the sound volume is over a given threshold, the selected motor action is carried out.
7) After a short delay, the algorithm checks whether the winning node has moved closer to the center of the map, and uses this to calculate a reward value for the reinforcement learning module.

Fig. 16 illustrates this procedure. The pipelined approach has the advantage of making experimenting with different processing paths much simpler. It also lends the approach to parallelising hardware and software implementations. In order to train the PLSOM a number of samples are recorded. We used white noise samples from 38 locations in front of the robot. The samples were recorded at 50 and 300 cm distance from the source, with 10° horizontal spacing. The training algorithm then presents a few seconds of random samples to the system for 10 000 training steps. Each sample is 256-dimensional, and a new sample is presented every 32 ms. This sensitises each node to sound from one direction and distance. The latter part of the training is done online, with the robot responding to actual sound from a stationary speaker in front of it. Initially the robot head is pointed in a random direction, and the training progresses until the robot keeps its head steadily pointed toward the speaker, at which time a new random direction is picked and the training continues.

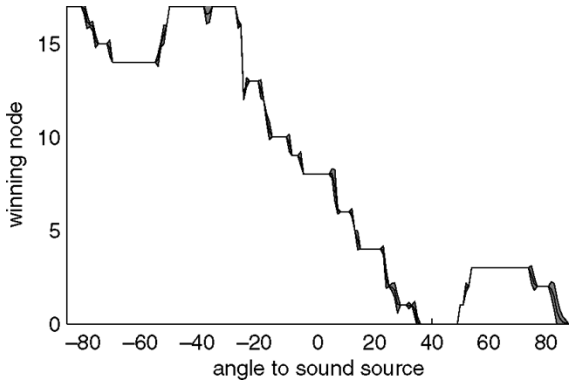


Fig. 17. High dimensional data. Average winning node versus actual angle using the PLSOM method. The grey area is one standard deviation.

3) Results: Our approach described previously consistently manages an accuracy of around 5°, which is comparable to human acuity. The precision was determined by keeping the robot stationary and registering the winning node of the PLSOM. We then moved the sound source horizontally until the winning node stabilised on one of the immediate neighbors of the initial winning cell, noting how much the source had to be moved. The relative accuracy of the method is demonstrated in Fig. 17. The graphs were generated using a set of recordings of white noise at a distance of 1 m. The PLSOM is, as we can see, almost free of deviation. This enables one to estimate the direction using a small number of samples, i.e., quickly.

B. Inverse Kinematics (IK)

Inverse kinematics (IK) is the problem of determining the joint parameters of a robotic limb for some given position. This problem is interesting in evaluating the PLSOM because it involves mapping between two spaces with wildly different topologies, a half-torus shaped space in Euclidean space and a roughly wedge-shaped space in joint parameter space.

1) Existing Methods: Robot control depends on IK but there are several problems associated with the existing methods. The Jacobian pseudo-inverse lets one solve the problem completely, but is computationally expensive, relatively complicated and unstable around singularities. The Jacobian transpose is faster and simpler, but not particularly accurate and does not move in a shortest path like the pseudo-inverse. Other methods, like cyclic coordinate descent [36], which solve the unstable singularity problem has been proposed. There have also been solutions of the IK problem using SOMs [2], which lets a SOM learn the joint angles for a number of points in space, and an approximation of the inverse Jacobian in the vicinity of each point. This gives good results with relatively few nodes but it relies on information being stored in the nodes of the SOM, rather than holistically in the network, leading each node to be more complicated than necessary. It is not clear whether using an approximated inverse Jacobian can lead to the same sort of instability around singularities as with the pseudo-inverse Jacobian. Even so, our method borrows heavily from the SOM approach and should be seen in relation to it.

2) Proposed Solution: We opted for a slightly different and, we believe, novel approach using the PLSOM. Each node maps

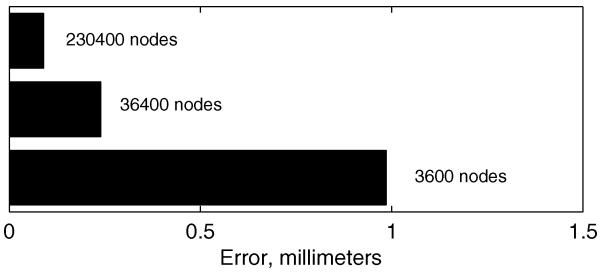


Fig. 18. Error after 500 iterations of some different sized PLSOMs.

a point in three-dimensional (3-D) space to a point in joint parameter space. The map is trained through generating a random joint configuration and updating the PLSOM weights with it. After the training is complete each node is labeled with the manipulator position that will result from applying the weight vector to the joint parameters. This allows the manipulator to be positioned in accordance with the following simple algorithm.

1) Select the node closest to the desired point in space by comparing node labels. This step can be greatly accelerated by starting the search at the last node used.
2) Select the neighbors of the node so that we can create three almost orthogonal vectors in 3-D space. This is trivial given the lattice structure of the PLSOM.
3) The three vectors in 3-D space are then orthogonalised using the Gram–Schmidt algorithm.
4) The analogous steps are carried out on the 3 corresponding vectors in joint parameter space.
5) The resulting vectors can now be used to interpolate the joint parameters.

3) Experiments and Results: The PLSOM solves the IK problem very quickly, as no iteration is necessary. One can input the desired target position and immediately get an approximation of the joint parameters that achieve this target. However, the level of precision depends on the number of nodes in the network. As a demonstration of the capabilities of the PLSOM method a 6-degree-of-freedom (DOF) robotic arm was programmed to play chess against itself. A PLSOM with 3600 nodes was trained in a half-torus shaped area in front of the robot covering a small chessboard and the surrounding table, 30000 training iterations are completed in less than 5 min on a low-end desktop PC. Even with the relatively few nodes the error is well below the mechanical error in the robotic arm. The robot is able to quickly and accurately pick and place the chess pieces, as shown in the short video clip in [37].

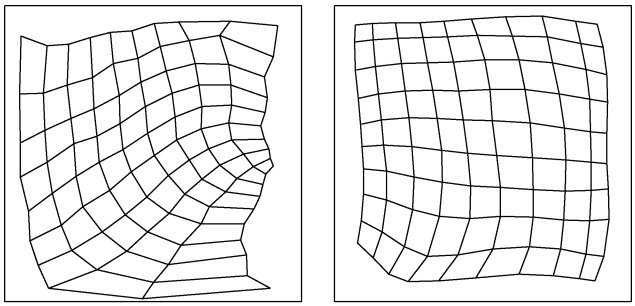
In order to assess the different IK methods, we performed a simulation wherein a 3-DOF robot arm is moved from one position to another. Each method is allowed 500 iterations to complete this and for each iteration the error is calculated. We performed this test with a PLSOM with 3600 nodes, a PLSOM with 36 400 nodes and a PLSOM with 230 400 nodes. We then repeated the experiment for 4 different target positions and averaged the error. The result is displayed in Fig. 18. As we can see from the graph, the PLSOMs accuracy is related to the number of nodes. It should be noted that training time is roughly $O(n)$ in number of nodes n, so increasing accuracy is not computationally expensive. Accuracy is slightly better than what is reported in [2], i.e., 0.02% error compared to 0.06% error, although this is not surprising given the large difference in number of nodes. The execution speed was measured and averaged over all the experiments—the difference in execution speed for the 3600 node network and the 230 400 node network is typically less than 2%, e.g., 29.52 and 29.98 μs on a low-end desktop PC.

C. Classification of the ISOLET Data Set

In order to test whether the PLSOM can handle very high-dimensional and clustered data, we selected the ISOLET [38] data set for analysis. The data set contains suitably processed recordings of 150 speakers saying the name for each of the letters in the alphabet, twice. The set is subdivided into the training set, containing 120 speakers, and the test set with the remaining 30 speakers. Algorithms seeking to solve this problem are trained on the training set and evaluated on the test set. The data is present as 617-dimensional real-valued vectors with elements in the $[-1, 1]$ range representing various preprocessed properties of the sound (see [38] for details). The best result reported without any further signal processing is 95.83% [39] and 95.9% [40], achieved with backpropagation. The k-nearest neighbor (k-NN) [41] methods achieve from 88.58% ($k = 1$) to 92.05% ($k = 5$, ambiguities resolved by decreasing k) but are slow due to the size of the training set which forces the computation of 6238 617-dimensional Euclidean distances for each classification. Classification of clustered data is troublesome with the PLSOM because it tries to approximate a low-dimensional manifold in the input space, in this case there may not be a manifold. Since the PLSOM is unsupervised there is no way to direct learning effort to one particular property of the input, the PLSOM tries to map them all. In the ISOLET set the letter is not the only property encoded in the data, there are all sorts of possible data that may or may not be present such as speaker, the speakers' age, gender, dialect, smoker/nonsmoker, and so on. Ideally the PLSOM would have one output dimension for each dimension of the embedded manifold, but this is impractical in this case. We, therefore, settled on a 3-D PLSOM with $20 \times 20 \times 20$ nodes. The PLSOM was trained with random samples from the training set for 100 000 iterations with neighborhood size $= 2$, then each node is labeled with the input it responds to. If a node responds to more than 1 input, a vote is performed and the node is labeled with the input it responds to most frequently. If no input gets thrice as many votes as other inputs, or if the node does not respond to any input, it is removed. The map is then used to classify the test set. This typically results in ca. 2800 remaining nodes, which we utilize as the reference vectors for k-NN classification. Thus, the PLSOM can be seen as a way of speeding up the k-NN algorithm by reducing the number of reference vectors. This does however come at a price of accuracy—it achieves 90.31% accuracy at $k = 5$.

VI. Discussion

As indicated above, the PLSOM emphasises maintaining topology over modeling the input density distribution. In many cases this may be a disadvantage, for example with highly clustered data. In other cases, however, this is exactly what is needed for a useful mapping of the input space. This happens when the input space is nonlinearly mapped to the space from



(a) (b)

Fig. 19. Three-beacon navigation mapping by the SOM and the PLSOM in the unit square. Both maps have a neighborhood size of 17. Beacons were positioned at (0.3,−0.3), (1.3,0.5) and (−0.5,0.8). The origin is in the upper left-hand corner. (a) SOM. (b) PLSOM.

which the inputs are selected. This is best illustrated by an example (based on [42]). A robot navigates a square area by measuring the angles between three uniquely identified beacons, giving a 3-D vector with a one-to-one relationship to the location of the robot in space. Unfortunately the density and topology of the embedded manifold is quite different from the input space, causing a large number of samples to fall in a few small areas of the 3-D input space where the curvature of the embedded Riemannian manifold is high. This causes the SOM to try and fit a corner of the map where there should be no corner, see Fig. 19(a). The density of the input space is correctly represented, but the topology of the embedded manifold (which is the interesting property in this case) is not. This can of course be corrected by selecting the samples carefully in the case of our example since we know the bidirectional mapping, but in a real application this would not be known. It would, therefore, be beneficial to find an approach that largely ignores the distribution of the input space and instead emphasises the topology of the embedded manifold. The PLSOM does this to a higher degree, see Fig. 19(b).

VII. Conclusion

We have addressed several problems with the popular SOM algorithm and proposed an algorithm that solves these; the PLSOM which provides a simplification of the overall application process, since it eliminates the problems of finding a suitable learning rate and annealing schemes. The PLSOM also reduces the training time. Flexibility and ordering of the map is facilitated and we have shown that the PLSOM can be successfully applied to a wide range of problems. Furthermore, the PLSOM is able to handle input probability distributions that lead to failure of the SOM, albeit this comes at the cost of lower correspondence between the input distribution and the weight density. While the PLSOM does not converge in the sense that a SOM will if the learning rate is allowed to reach 0, the same effect can be achieved by simply not performing weight updates after a number of inputs. All this is achieved without inducing a significant computation time increase or memory overhead. Finally we have shown (see Appendix I) that the PLSOM is guaranteed to achieve ordering under certain conditions.

Appendix I
Proof of Gfdx Ordering of a PLSOM With T Nodes and 1-D Input and Output

This section will present a proof of guaranteed ordering in a special case of the PLSOM. We start out by establishing some lemmas necessary for the proof, then we examine the proof and finally we speculate implications of the proof. For all these proofs we assume that

$$\epsilon \leq 1 \tag{18}$$

where ϵ is the normalized distance from the input to the weight of the winning (closest) node

$$h_{c,i} \leq 1 \tag{19}$$

where $h_{c,i}$ is a neighborhood function which depends on ϵ and the distance in output space between the winning node c and the current node i. Please see Section II for a discussion of the neighborhood function. $c = i \Rightarrow h_{c,i} = 1$ and $h_{c,i}$ is monotonously decreasing with increasing distance from c

$$h_{c,n} < h_{c,n+1} \tag{20}$$

which implies that w_{n+1} is closer to the input than w_n, and will, hence, receive a larger scaling from the neighborhood function. In the following an ordered map will denote a map where all the nodes are monotonously increasing or decreasing, which means either 21 or 22 is true

$$w_n < w_{n+1} \qquad \forall n \tag{21}$$

$$w_n > w_{n+1} \qquad \forall n. \tag{22}$$

Any other map will be called unordered.

Lemma 1: The weights of a node cannot overshoot the input, i.e., the weights of a node cannot move from one side of an input to the other as a result of that input.

Proof: Assume[1] that there is an input x, and a node i with weight w_i. The amount of update to w_n, Δw_i is equal to $\epsilon h_{c,i}[x - w_i]$ as before. Since $\epsilon \leq 1$ and $h_{c,i} \leq 1$, it is clear that $|\Delta w_i| \leq |x - w_i|$. Also, $|\Delta w_i| = |x - w_i|$ is only true where $\epsilon = 1 \wedge h_{c,i} = 1$, which can only hold for the winning node c. This proof also applies to the standard SOM where $\alpha(t) \leq 1$.

Lemma 2: There exists no input that can turn an ordered 1-input dimension and 1-output dimension map into an unordered one.

Proof: Proof by contradiction. Assume that w_n and w_{n+1} are weights of an ordered one-dimensional (1-D) map, and $w_n < w_{n+1}$. We will prove that no input $x \geq w_{n+1}$ can move w_n past w_{n+1} (It is easy to see that the converse has to be true for $x \leq w_n$). For w_n to move past w_{n+1} and making the map unordered, the update of node n must be greater than or equal to the distance between node n and node $n+1$, plus the update to node $n+1$, so (23) has to be true

$$\Delta w_n \geq w_{n+1} - w_n + \Delta w_{n+1}. \tag{23}$$

[1]Note that for this section we disregard the (t) part of the notation, as it is implied.

Remember that $\Delta w_n = \epsilon h_{c,n}(x - w_n)$ and that $x \geq w_{n+1} > w_n$, which gives us (24)

$$\epsilon h_{c,n}(x - w_{n+1}) + \epsilon h_{c,n}(w_{n+1} - w_n) \geq w_{n+1} - w_n + \epsilon h_{c,n+1}(x - w_{n+1}). \quad (24)$$

It is clear that, because of the premises, (24) cannot be true. This is a restatement of the proof that the ordered states are absorbing sets, see [43]

That leaves the cases where $w_n < x < w_{n+1}$. Since, according to lemma 1, only one node (the winning node) can reach x and no node can overshoot x, it follows that the nodes must be on the same side of x after the weight update as before. Therefore, they cannot become unordered.

Note that the lemma 1 and 2 holds for PLSOMs with any number of nodes, as long as there is one input and one output dimension. This proof is very similar to the one given for the SOM by Kohonen [44].

Lemma 3: In the special case where $w_i = a, \forall i$ and some value a, any input other than a will result in an ordered map.

Proof: Since all nodes have the same distance to the input, the winning node will automatically be the first one, w_0. Again, since all nodes are the same distance from the input, the amount each node is updated is determined solely on the lattice distance from the winning node. Therefore, w_0 will move the most, w_1 a little less and w_2 even less and so on—resulting in an ordered map.

Lemma 4: A 1-D PLSOM with three nodes will always reach an ordered state given a sufficiently large number of uniformly distributed inputs.

Proof: The proof is computer-assisted, here we give an outline of the procedure for calculating it.

In short, the proof is as follows.

1) Calculate a scalar field in expressing how much closer the weights are to an attractor point in ordered space after an update, where positive values indicate that the weights have moved away from the attractor.
2) Calculate the gradient of the scalar field.
3) Calculate the upper bound of the gradient.
4) Given the upper bound of the gradient we calculate the size of a sample grid.
5) Given the upper bound of the gradient and the expected update at each sample point, the expected update must point toward the attractor in the vicinity of the sample point. If this holds for all sample points, it holds for the whole subspace of unordered weights.

The weights of the three PLSOM nodes are denoted w_0, w_1, w_2. The conditions under which the proof is calculated are as follows.

- Uniformly distributed input in the range [0,1].
- $w_0 \leq w_2 \leq w_1$.
- $w_0 < w_1$.
- $0 \leq w_0 < 1$.
- $0 \leq w_2 \leq 1$.
- $0 < w_1 \leq 1$.
- Linear neighborhood function, for simplicity.

The weights w_0, w_1 and w_2 can be seen as coordinates in a 3-D space, where all possible configurations fill the unit cube. The subspace spanned by the constraints above are denoted U, see Fig. 20. This subspace represents the only way in which a 1-D three-node map can be unordered—all other unordered states are inversions or mirrors of this state, see Fig. 21. An ordered map fulfils one of the following configurations: $w_0 < w_1 < w_2$ and $w_0 > w_1 > w_2$.The ordered subspace fills the volume drawn in Fig. 22. Now on to the proof proper.

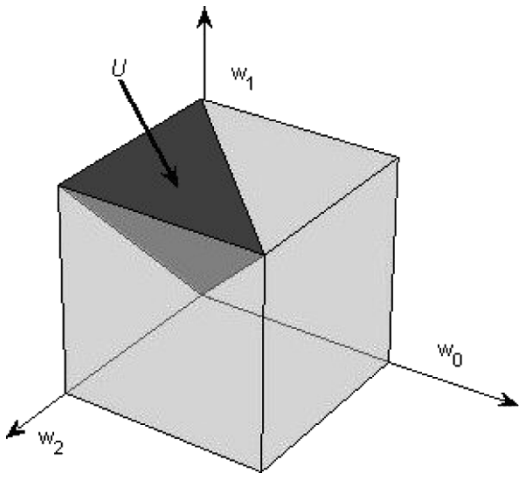


Fig. 20. Unordered subspace U. All other unordered states are mirrors or inversions of states in this subspace.

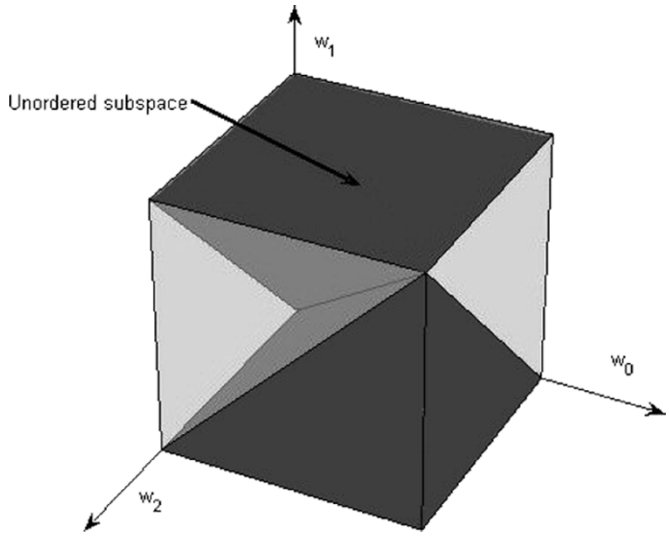


Fig. 21. All unordered states in the volume of all possible states.

We introduce the concept of the expected update vector: Given a weight configuration and an input probability density function we can compute the distance and direction that a node is most likely to move in and is given

$$\vec{u} = [u_0, u_1, u_2] \quad (25)$$

where $\vec{u}$ is the expected update vector, its elements given by

$$u_n = \frac{w_0 + w_2}{2} \int_0^{\frac{w_0+w_2}{2}} f_n(0)dx + \frac{w_1 - w_0}{2} \int_{\frac{w_0+w_2}{2}}^{\frac{w_2+w_1}{2}} f_n(2)dx + \left(1 - \frac{w_2 + w_1}{2}\right) \int_{\frac{w_2+w_1}{2}}^{1} f_n(1)dx \quad (26)$$

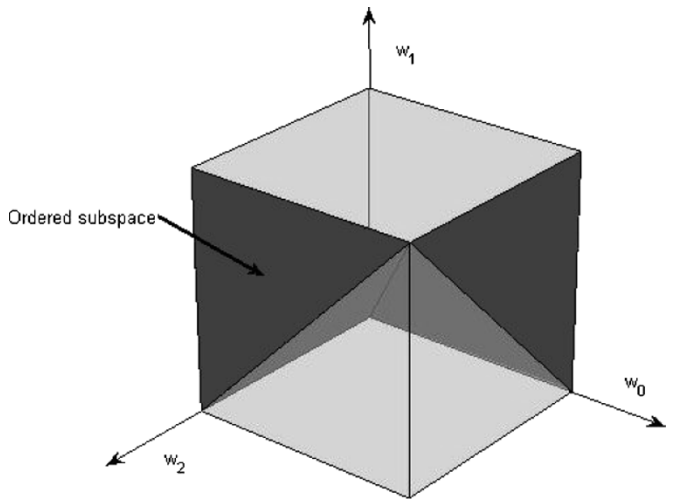


Fig. 22. All ordered states in the volume of all possible states.

where $f_n(c)$ is the expected update of node n given c as the winning node. As mentioned above, we use a simplified version of $f_n(c)$ to facilitate integration

$$f_n(c) = \frac{|x - w_c|}{r}\left(1 - \frac{|c - n|}{\beta}\right)(x - w_n) \qquad (27)$$

where r is the normalizing variable and β is the neighborhood size. For simplicity we set $r = 1$ and $\beta = 2$. x is the input, uniformly distributed in the [0,1] interval.

As mentioned above, the three input weights of the nodes, w_0, w_1 and w_2 can be seen as coordinates in a 3-D Euclidean space, of which U is a subspace.

Every point $\vec{w}$ in this subspace is associated with an expected update $\vec{u}$: The position before an input is represented by $\vec{w}$, and the most likely position after a uniformly distributed random input is $\vec{w} + \vec{u}$.

Now we introduce a point denoted $\vec{t}$ in the *ordered* subspace of the unit cube, which is the attractor in the dynamic system in U. In other words, all the expected weight updates in U will bring the weight vectors closer to the attractor, and, hence, closer to the ordered subspace

$$||\vec{w} + \vec{u} - \vec{t}||_1 - ||\vec{w} - \vec{t}||_1 < 0,\ w \in U \qquad (28)$$

where $||.||_1$ is the L^1-norm or Manhattan distance. The L^1-norm was chosen because it produces a simpler expression than the L^2-norm. $\vec{t}$ has been found empirically to be close to [377/1000, 121/200, 7/10], the exact location is not important to this proof. Equation (28) defines a 3-D scalar field and in order to prove negativity we compute the upper bound of the length of the gradient, 16.5. With this estimated upper bound we must check that no point is further away from a sample point than $1.333 * 10^{-4}$, and that no sample point has a value greater than $d = -2.2 * 10^{-3}$. This gives

$$s = \sqrt{\frac{4d^2}{3}} \qquad (29)$$

where s is the spacing of the 3-D grid of sample points, $1.539\,59 * 10^{-4}$. This equals roughly $4.57 * 10^{10}$ sample points

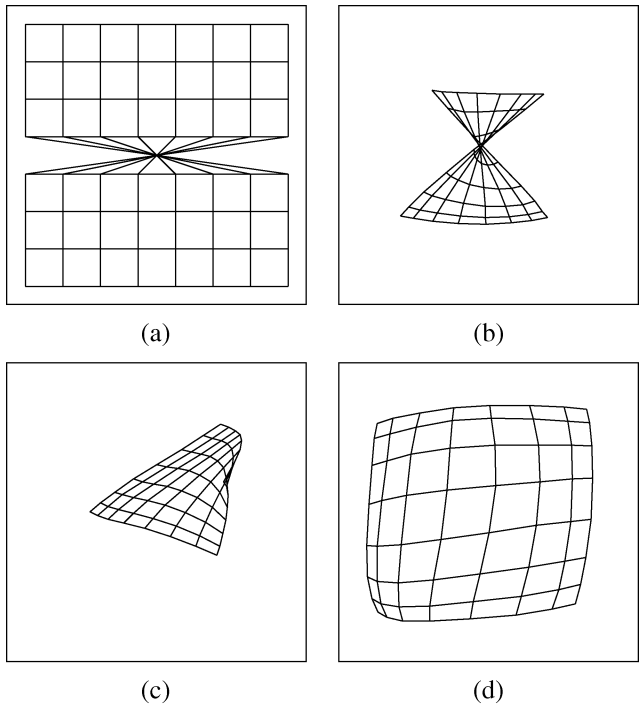


Fig. 23. Evolution of the weight positions of a 64-node 2-D PLSOM initialized to a difficult position. Neighborhood size is 11, minimum neighborhood size is 0. To simulate what will happen if this configuration appears late in training, we force an r value of 0.65. (a) Initial state. (b) After 400 inputs. (c) After 480 inputs. (d) After 650 inputs.

to check, another reason for choosing the simpler L^1-norm. The necessary calculations are easily performed by a low-end desktop computer in less than 12 hours. Since the distance from the weight position to the attractor is steadily diminishing, it follows that the weight position will, given enough consecutive inputs, come close enough to the attractor to reach ordered space.

Whether this proof is extensible to networks with more than three nodes and more than 1-D input is at this point uncertain, but the image sequence in [Figs. 23(a)–23(d)] certainly suggests the possibility.

Kohonen [44] mentions a proof (see [43] and [45]) of ordering of a simplified SOM based on the probability of an ordering input happening and an infinite number of inputs. This proof in essence relies on the fact that if there is a *sequence of inputs* such that the map will become ordered and one generates a sufficiently large number of inputs the probability of encountering the ordering sequence of inputs approach 1. The proof just presented here establishes that for any configuration, the expected update is in the direction of ordering for *any single input*. It also shows the existence of an ordered attractor for the dynamical system without having to satisfy the Robbins–Monro [46] condition.

Conjecture: Any 1-D PLSOM where only the immediate neighbors of the winning node are updated can be seen as a chain of three-node networks, where each subnetwork is guaranteed to become ordered, therefore, the whole network will become ordered. This is similar to the proof given in [44] for the SOM, albeit the authors are not confident enough that it also applies to the PLSOM to posit it as more than a conjecture.

ACKNOWLEDGMENT

The authors would like to thank Dr. F. Maire of the Smart Devices Laboratory, Queensland University of Technology, Australia, and Dr. G. Wyeth of the Division of Complex and Intelligent Systems, University of Queensland, Australia, for their valuable input.

REFERENCES

[1] T. Kohonen, "The self-organizing map," *Proc. IEEE*, vol. 78, no. 9, pp. 1464–1480, Sep. 1990.
[2] H. Ritter, T. Martinetz, and K. Schulten, *Neural Comput. and Self-Organizing Maps—An Introduction*. Reading, MA: Addison-Wesley, 1992.
[3] C. M. Bishop, M. Svensén, and C. K. I. Williams, "GTM: A principled alternative to the self-organizing map," *Adv. Neural Inf. Process. Syst.*, vol. 1, no. 9, pp. 354–360, 1997.
[4] ——, "GTM: The generative topographic mapping," *Neural Comput.*, vol. 10, no. 1, pp. 215–235, 1998.
[5] A. Vellido, W. El-Deredy, and P. J. G. Lisboa, "Selective smoothing of the generative topographic mapping," *IEEE Trans. Neural Netw.*, vol. 14, no. 4, pp. 847–852, Jul. 2003.
[6] J. H. Kaas, "Plasticity of sensory and motor maps in adult mammals," *Annu. Rev. Neurosci.*, vol. 14, pp. 137–167, Mar. 1991.
[7] J. Göppert and W. Rosenstiel, "Varying cooperation in SOM for improved function approximation," in *IEEE Int. Conf. Neural Netw.*, vol. 1, 1996, pp. 1–6.
[8] R. Lang and K. Warwick, "The plastic self organizing map," in *Proc. 2002 Int. Joint Conf. Neural Netw.*, vol. 1, 2002, pp. 727–732.
[9] B. Fritzke, "Growing cell structures—a self-organizing network for unsupervised and supervised learning," *Neural Netw.*, vol. 7, no. 9, pp. 1441–1460, 1994.
[10] ——, *A Growing Neural Gas Network Learns Topologies*. Cambridge, MA: MIT Press, 1995, pp. 625–632.
[11] R. Iglesias and S. Barro, "SOAN: self organizing with adaptive neighborhood neural network," in *Proc. IWANN*, 1999, pp. 591–600.
[12] H. Shah-Hosseini and R. Safabakhsh, "TASOM: the time adaptive self-organizing map," *Proc. ITCC*, pp. 422–427, 2000.
[13] ——, "TASOM: a new time adaptive self-organizing map," *IEEE Trans. Syst., Man, Cybern. B*, vol. 33, no. 2, pp. 271–282, Feb. 2003.
[14] K. Haese, "Kalman filter implementation of self-organizing feature maps," *Neural Comput.*, vol. 11, no. 5, pp. 1211–1233, 1999.
[15] K. Haese and G. J. Goodhill, "Auto-SOM: recursive parameter estimation for guidance of self-organizing feature maps," *Neural Comput.*, vol. 13, no. 3, pp. 595–619, 2001.
[16] J. A. Starzyk, Z. Zhu, and T.-H. Liu, "Self-organizing learning array," *IEEE Trans. Neural Netw.*, vol. 16, no. 2, pp. 355–363, Feb. 2005.
[17] T. Kwok and K. A. Smith, "A noisy self-organizing neural network with bifurcation dynamics for combinatorial optimization," *IEEE Trans. Neural Netw.*, vol. 15, no. 1, pp. 84–98, Jan. 2004.
[18] E. Berglund and J. Sitte, "The parameter-less SOM algorithm," in *Proc. ANZIIS*, 2003, pp. 159–164.
[19] A. Campbell, E. Berglund, and A. Streit, "Graphics hardware implementation of the parameter-Less self-organizing map," in *Proc. IDEAL*, 2005, pp. 343–350.
[20] A. King and S. Carlile, *Neural Coding for Auditory Space*. Cambridge, MA: Mit Press, 1995.
[21] D. Nandy and J. Ben-Arie, *Auditory Localization Using Spectral Information*. New York: Academic, 1995.
[22] W. M. Hartmann, "How we localize sound," *Phys. Today*, vol. 1, no. Nov., pp. 23–29, 1999.
[23] R. S. Sutton, Ed., *Reinforcement learning*. Norwell, MA: Kluwer, 1992.
[24] J. Huang, N. Ohnishi, and N. Sugie, "A biometric system for localization and separation of multiple sound sources," *IEEE Trans. Instrum. Meas.*, vol. 44, no. 3, pp. 733–738, Mar. 1995.
[25] ——, "Building ears for robots: sound localization and separation," *Artif. Life Robot.*, vol. 1, no. 4, pp. 157–163, 1997.
[26] D. Rabinkin, R. Renomeron, A. Dahl, J. French, J. Flanagan, and M. Bianchi, "A DSP implementation of source location using microphone arrays," *Proc. SPIE*, vol. 2846, pp. 88–99, 1996.
[27] T. Nakatani, H. G. Okuno, and T. Kawabata, "Auditory stream segregation in auditory scene analysis with a multi-agent system," in *Proc. AAAI4*, 1994, pp. 100–107.
[28] H. Kitano, H. G. Okuno, K. Nakadai, T. Matsui, K. Hidai, and T. Lourens. (2002) SIG, The Humanoid. [Online]. Available: http://www.symbio.jst.go.jp/symbio/SIG/
[29] K. Nakadai, T. Lourens, H. G. Okuno, and H. Kitano, "Active audition for humanoid," in *Proc. AAAI*, 2000, pp. 832–839.
[30] T. Lourens, K. Nakadai, H. G. Okuno, and H. Kitano, "Humanoid active audition system," presented at the *IEEE-RAS Int. Conf. Humanoid Robots*, 2000, CD-ROM.
[31] K. Nakadai, H. Okuno, and H. Kitano, "Realtime sound source localization and separation for robot audition," in *Proc. IEEE Int. Conf. Spoken Language Process.*, 2002, pp. 193–196.
[32] K. Nakadai, H. G. Okuno, and H. Kitano, "Robot recognizes three simultaneous speech by active audition," in *Proc. ICRA*, vol. 1, Sep. 2003, pp. 398–405.
[33] M. Rucci, G. Edelman, and J. Wray, "Adaptation of orienting behavior: from the barn owl to a robotic system," *IEEE Trans. Robot. Autom.*, vol. 15, no. 1, pp. 96–110, Feb. 1999.
[34] H. Nakashima, T. Mukai, and N. Ohnishi, "Self-Organization of a sound source localization robot by perceptual cycle," in *Proc. ICONIP*, Nov. 2002, pp. 834–838.
[35] M. P. Zwiers, A. J. V. Ostal, and J. R. M. Cruysberg, "Two-dimensional sound-localization behavior of early-blind humans," *Exp. Brain Res.*, vol. 140, pp. 206–222, 2001.
[36] L.-C. Wang and C. Chen, "A combined optimization method for solving the inverse kinematics problems of mechanical manipulators," *IEEE Trans. Robot. Autom.*, vol. 7, no. 4, pp. 489–499, Aug. 1991.
[37] E. Berglund. (2004) PLSOM-Controlled 6 degree of freedom robotic arm playing chess. [Online]. Available: http://www.itee.uq.edu.au/~berglund/robochess.mov
[38] R. A. Cole, Y. K. Muthusamy, and M. Fanty, "The ISOLET spoken letter database," Oregon Grad. Inst., Beaverton, Tech. Rep. 90-004, 1990.
[39] T. G. Dietterich and G. Bakiri, "Error-correcting output codes: a general method for improving multiclass inductive learning programs," in *Proc. AAAI*, 1991, pp. 572–577.
[40] M. Fanty and R. Cole, "Spoken letter recognition," in *Proc. 1990 Conf. Adv. Neural Inf. Process. Syst. 3*, 1990, pp. 220–226.
[41] E. Fix and J. Hodges, "Discriminatory analysis, non-parametric discrimination," USAF Sch. Aviation Medicine, Randolf Field, TX, Tech. Rep. 4, 1951.
[42] N. Keeratipranon and F. Maire, "Bearing similarity measures for self-organizing feature maps," in *Proc. IDEAL*, 2005, pp. 286–293.
[43] M. Cottrell, J. C. Fort, and G. Pagès, "Theoretical aspects of the SOM algorithm," *Neurocomput.*, vol. 21, pp. 119–138, Nov. 1998.
[44] T. Kohonen, *Self-Organization and Associative Memory*, 3rd ed. New York: Springer-Verlag, 1989.
[45] M. Cottrell and J. Fort, "Etude d'un algorithme d'auto-organization," *Annal. Inst. Henri Poincare (B) Prob. stat.*, vol. 23, no. 1, pp. 1–20, 1987.
[46] H. Robbins and S. Muonro, "A stochastic approximation method," *Annal. Math. Statist.*, vol. 22, pp. 400–407, 1951.

Erik Berglund received the B.S. degree in computer engineering from Østfold University College, Sarpsborg, Norway, in 2000. He is currently working toward the Ph.D. degree at the University of Queensland, St. Lucia, Australia.

His research interests include computational intelligence, robotics, and implicit data processing.

Joaquin Sitte received the Licenciado degree in physics from the Universidad Central de Venezuela, Caracas, Venezuela, in 1968, and the Ph.D. degree in quantum chemistry from Uppsala University, Uppsala, Sweden, in 1974.

Since 1986, he has been with the Queensland University of Technology, Brisbane, Australia, where he is currently an Associate Professor at the School of Software Engineering and Data Communications, where he leads the Smart Devices Laboratory. Until 1985, he was an Associate Professor at the Universidad de Los Andes, Merida, Venezuela, and Head of the Surface Physics Research Group. Since 1986, he is on the faculty of Queensland University of Technology. Research interests include the use of neural networks robotics.