The Parameter-Less SOM algorithm

Erik Berglund
Smart Devices Lab, QUT
e.berglund@student.qut.edu.au

Dr. Joaquin Sitte
Smart Devices Lab, QUT
j.sitte@qut.edu.au

Abstract

One of the biggest problems facing practical applications of Self-Organising Maps (SOM) is their dependence on the learning rate, the size of the neighbourhood function and the decrease of these parameters as training progresses, all of which have to be selected without firm theoretical guidance. This paper introduces a simple modification to the SOM that completely eliminates the learning rate, the decrease of the learning rate and the decrease of the neighbourhood size. This is done by making the learning rate and neighbourhood size dependant on a variable calculated from the internal state of the SOM, rather than on externally applied variables.

Keywords: Self-Organising Map, SOM, Parameter-Less, PLSOM, learning rate, neural network, neighbourhood function, neighbourhood size, unsupervised learning.

1. Introduction

Training of many Artificial Neural Networks, including the SOM [6], depends on learning rates to converge. The learning rate is a variable that governs how much the weights of a network are altered in response to an input. For SOMs the learning rate must be decreased according to some algorithm as learning progresses, this process is referred to as annealing in this document. The update of the weight vector in the standard SOM depends on the error of that particular weight vector, but to make sure the SOM converges towards a stable state a learning rate is required. The learning rate is large in the beginning of the training, when the map is unordered and fits the input space poorly, and small the end of the training when there is only need for small changes to the map.

Learning rates present a two-sided problem to users and researchers of Self-Organising Maps:

1. There exists no theoretical way of estimating the starting learning rate, nor the rate at which it should decline (the annealing scheme). There exist general guidelines and heuristics, but in the last instance the learning rate has to be adapted to a specific task by empirical methods.
2. Explaining the learning rate from the perspective of the SOM as a model of biological neural networks is difficult: While the human brain loses some of its flexibility over time, learning is much more rapid than the reduction of flexibility and learning is still possible after the reduction. Even complex neural networks like mammalian brains are capable of drastically changing their mapping if the input changes suddenly, such as after an amputation [5].

The choice of learning rate and annealing schemes greatly affects the network's ability to reach a stable and well-ordered state and the time it takes to reach this state [8]. This led to the search for a method by which the optimal learning rate and decay can be determined with mathematical certainty given a specific problem, for example [7]. Unfortunately, this has not produced a universally valid, simple and computationally effective algorithm. Some previous works include the Generative Topographic Mapping (GTM) algorithm [1, 2] which eliminates the neighbourhood function entirely, but relies on other variables and choice of prior distribution and basis functions. The Maximum Entropy learning Rule (MER) algorithm [4] achieves global ordering without use of a neighbourhood at all, but is slower and still relies on a learning rate. The Growing Neural Gas (GNG) algorithm [3] is similar to the algorithm proposed in this paper in the way it computes the error and in that it eliminates the annealing schemes, but instead of scaling neighbourhood size and learning rate with the error, it uses the accumulated error per neuron to determine where to insert a new cell in the network, in addition it introduces new variables that must be pre-determined with no firm theoretical grounding. Many supervised learning algorithms feature the idea of scaling the learning rate according to the error (Newton's method could be seen as an early example of this) - however the error in a supervised learning scheme is readily available, which is not the case in unsupervised learning schemes like the SOM. This still leaves us with a SOM algorithm that relies on empirical testing to determine the annealing scheme - a poor choice of learning rate, learning rate annealing scheme or neighbourhood size annealing scheme can result in knotting, folding, a poor input space fit or later inputs destroying what was learned by earlier inputs.

The solution we propose in this paper is to let the scaling of the weight vector update function and/or the size of the neighbourhood depend on internal conditions in the map, instead of or in combination with externally enforced scaling variables such as the learning rate. The internal condition we selected for scaling these variables is the least error ϵ, i.e. the normalized euclidean distance from the input to the closest weight vector, see Section 3.1. It is intuitive that if this variable is large, the map needs to change more to accommodate future inputs of this class, but if it is small the fit is already good and there is no need for large alterations of the map. In this paper we will examine two variants of the standard SOM algorithm, we will demonstrate the Parameter-Less SOM (PLSOM) and compare its performance to the performance of a widely used SOM algorithm, namely the variant implemented in the Matlab neural networking package, and discuss some of their relative merits.

2. Background

2.1. The standard SOM algorithm

The SOM we will be modifying in this paper is the gaussian-neighbourhood, euclidean distance, rectangular topology SOM, given by equations (2)-(5). The algorithm is, in brief, as follows: An input $x(t)$ is presented to the network at epoch (or timestep, iteration) t. The 'winning neuron', i.e. the neuron with the weight vector that most closely match the input, is selected using equation (1).

$$c = \arg\min_i(||x(t) - w_i(t)||_2) \quad (1)$$

$w_i(t)$ is the weight vector of cell i at epoch t. $||.||_2$ denotes the l_2-norm or n-dimensional Euclidian distance. There are other ways of computing the distance than the Euclidian distance given in equation (1) that can be used in the SOM, for example Manhattan or link distance. The weight update is calculated using equations (2) and (3).

$$w_i(t+1) = w_i(t) + \alpha(t)h_{ci}(t)[x(t) - w_i(t)] \quad (2)$$

$$h_{ci}(t) = e^{\frac{-d(i,c)^2}{\beta(t)^2}} \quad (3)$$

$h_{ci}(t)$ is referred to as the neighbourhood function, and is a scaling function centred on the winning cell c decreasing in all directions from it. $d(i, c)$ is the euclidean distance from cell i to the winning cell c. $\alpha(t)$ is the learning rate at epoch t, $\beta(t)$ is the neighbourhood size at epoch t.

Lastly the learning rates are decreased in accordance with the annealing scheme. We have chosen the annealing scheme given by equations (4) and (5) for the decrease of the learning rate and the neighbourhood size, respectively.

$$\alpha(t+1) = \alpha(t)\delta_\alpha, \qquad \delta_\alpha < 1 \quad (4)$$

$$\beta(t+1) = \beta(t)\delta_\beta, \qquad \delta_\beta < 1 \quad (5)$$

Here δ_β and δ_α are scaling constants determined beforehand, typically around 0.9999.

These steps are repeated until some preset condition is met, usually after a give number of epochs or when some measurement of error reaches a certain level.

It should be noted that one of the most widely used implementations of the SOM algorithm differs in some respects from this. The overall algorithm is the same, but the annealing scheme is different: It is divided into two phases, phase 1 (ordering phase) and phase 2 (tuning phase). Matlab also uses a step-based neighbourhood function, equation (6). Equations (7) and (8) govern the annealing by reducing the learning rate and the neighbourhood size, respectively.

$$h_{ci}(t) = \begin{cases} 1 & \text{if } d(i,c) = 0 \\ 0.5 & \text{if } d(i,c) < \beta(t) \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

$$\alpha(t+1) = \begin{cases} \alpha(0) + ((\alpha(0) - \alpha(\tau))(1 - \frac{t}{\tau})) & \text{if } t < \tau \\ \alpha(\tau)\frac{\tau}{t} & \text{otherwise} \end{cases} \quad (7)$$

$$\beta(t+1) = \begin{cases} \beta(0) + ((\beta(0) - \beta(\tau))(1 - \frac{t}{\tau})) & \text{if } t < \tau \\ \beta(\tau) & \text{otherwise} \end{cases} \quad (8)$$

τ is the number of epochs in phase 1 (default: 1000), $\alpha(0)$ is predefined (default: 0.9) and $\beta(0)$ is calculated based on the network size, for the two dimensional case this is given by (9). $\alpha(\tau)$ is predefined (default: 0.02) and $\beta(\tau)$ is 1.00001.

$$\beta(0) = \sqrt{(w-1)^2 + (h-1)^2} \quad (9)$$

w and h is the number of cells in the vertical and the horizontal directions, respectively.

The difference between these two variants of the SOM is that while the Matlab version achieves basic ordering quicker, it does not fit the data as well as the other variant.

3. The Parameter-Less SOM

The PLSOM relies on the idea that the learning rate and neighbourhood size should not vary according to the iteration number, but rather vary according to how well the map represents the topology of the input space.

3.1. Algorithm

The scaling variable depending on how good the fit of the weight vector of the winning neuron is to the last input, ϵ, is defined in equations (10) and (11).

$$\epsilon(t) = \frac{||x(t) - w_c(t)||_2}{\rho(t)} \tag{10}$$

$$\begin{aligned} \rho(t) &= \max(||x(t) - w_c(t)||_2, \rho(t-1)), \\ \rho(0) &= ||x(0) - w_c(0)||_2 \end{aligned} \tag{11}$$

$\epsilon(t)$ 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.

The algorithm for the PLSOM uses a neighbourhood size determined by ϵ, thus replacing the equation governing the annealing of the neighbourhood with $\beta(t) = constant \ \ \forall t$. Furthermore it uses equation (14) for weight updates and equation (13) for the neighbourhood function.

$$\Theta(t) = \beta(t)\epsilon(t), \qquad \Theta(t) >= \theta_{min} \tag{12}$$

$$h_{ci}(t) = e^{\frac{-d(i,c)^2}{\Theta(t)^2}} \tag{13}$$

$$w_i(t+1) = w_i(t) + \epsilon(t)h_{ci}(t)[x(t) - w_i(t)] \tag{14}$$

As we can see from equation (14) the learning rate $\alpha(t)$ is now completely eliminated, replaced by $\epsilon(t)$. All examples in this paper are based on 2-dimensional input spaces and maps. Since the Euclidean distance can be calculated in n-dimensional space it is likely that the PLSOM algorithm can be adapted to any number of input- and output dimensions.

3.2. Advantages

The PLSOM completely eliminates the selection of the learning rate, the annealing rate and annealing scheme of the learning rate and the neighbourhood 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.

3.2.1 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, 2 dimensional 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.

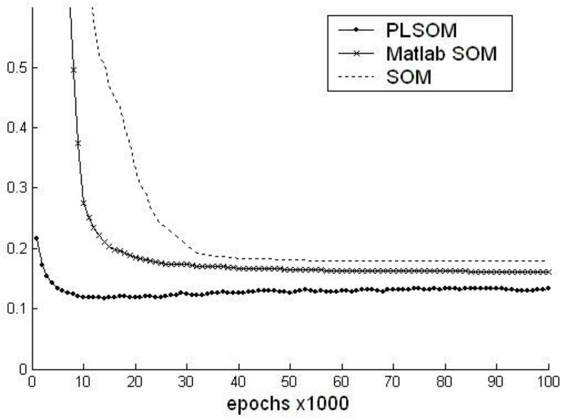

Figure 1. Graph of the decrease of uncovered space as training progresses for the PLSOM, the SOM and the Matlab SOM implementation. Note the quick expansion of the PLSOM and that it consistently covers a larger area than the SOM variants.

In comparing the two SOM implementations we used 3 separate metrics, 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 neighbouring neurons.

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 maintain the lead throughout the simulation (figure 1). Please note that this metric will be misleading in situations where cells are overlapping.

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. 24000 epochs the SOM surpasses the PLSOM. After 100000 epochs the difference is still small, however. See figure 2.

Deviation of cell size We calculate the absolute mean deviation of the cell size and divide it by the average cell

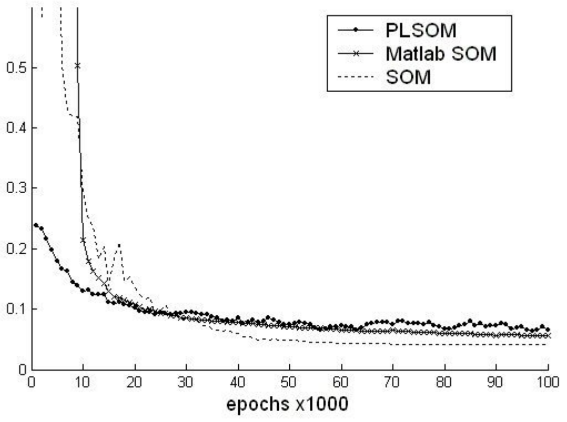

Figure 2. Graph of the average skew for the PLSOM, the SOM and the Matlab SOM implementation. For the first 24000 iterations the PLSOM is more ordered, before the SOM variants narrowly overtakes it.

size to get an idea of how much the cells differ in relative size. Here the SOM is superior to the PLSOM after ca. 10000 epochs, mainly because of the flattened edge cells of the PLSOM, see figure 3.

If we ignore the cells along the edge, the picture is quite different: the PLSOM outperforms the SOM with a narrow margin, see figure 4.

All experiments was carried out using Matlab and the JRobot package [9]. Thanks to Dr. Mark Hale for his contribution to JRobot, and to Dr. Steffen Log.

3.2.2 Plasticity preservation

The illustrations in this section shows 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 figures 5 and 6, where a SOM has been presented with pseudorandom, uniformly distributed 2-dimensional data vectors in the $[0, 0.5]$ range for 50000 iterations. Thereafter the SOM was presented with 20000 pseudorandom, uniformly distributed 2-dimensional 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 to the new input range almost immediately, as seen in figures 7 and 8.

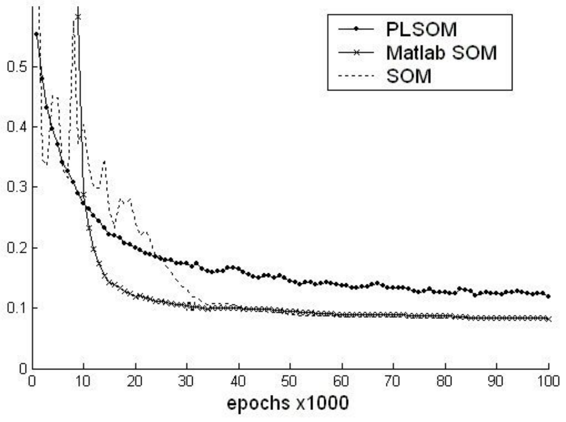

Figure 3. Graph of the absolute mean deviation of cell size for the PLSOM, the SOM and the Matlab SOM. The PLSOM is more regular up until ca. epoch 10000.

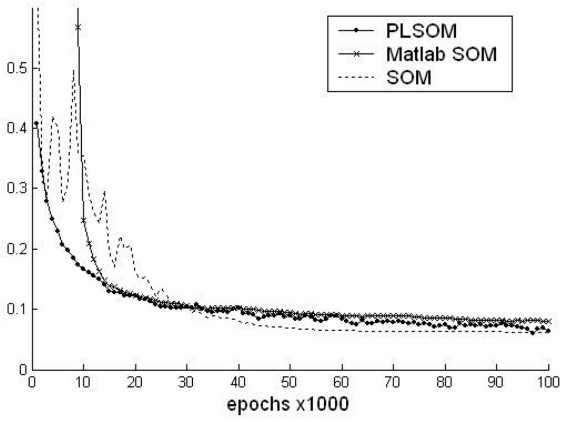

Figure 4. Graph of the absolute mean deviation of cell size for the PLSOM, the SOM and the Matlab SOM, excluding the edge cells. Compare to figure3. The PLSOM outperforms the Matlab SOM in both adaption time and accuracy, and the SOM needs until ca. epoch 30000 to reach the same level of ordering.

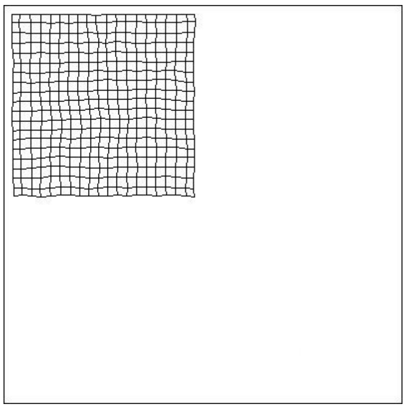

Figure 5. SOM weight vector position in input space after training for 50000 iterations with uniformly distributed pseudorandom 2-dimensional input, ranging from 0 to 0.5.

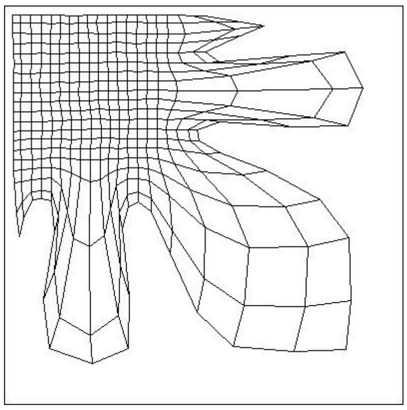

Figure 6. Same SOM as in figure 5 after 20000 further training steps with inputs ranging from 0 to 1.0.

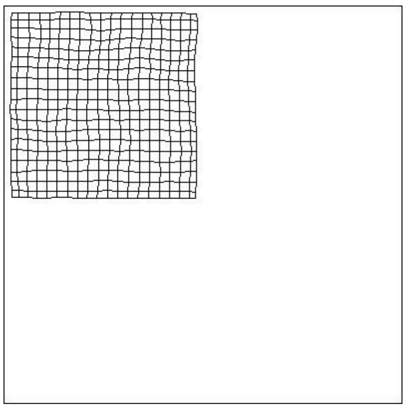

Figure 7. PLSOM weight vector position in input space after training for 50000 iterations with uniformly distributed pseudorandom 2-dimensional input, ranging from 0 to 0.5.

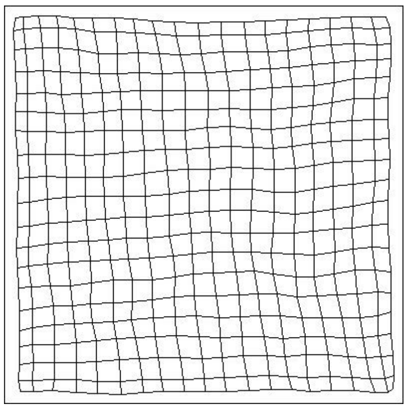

Figure 8. Same PLSOM as in figure 7 after 20000 further training steps with inputs ranging from 0 to 1.0. Note the difference between this figure and figure 6.

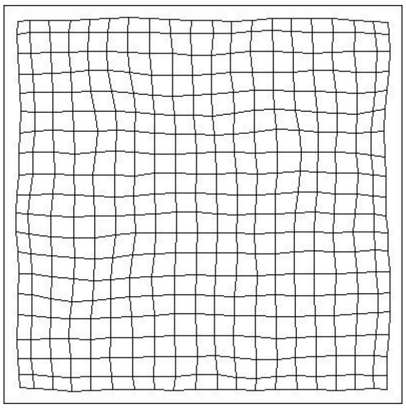

Figure 9. PLSOM weight vector position in input space after training for 50000 iterations with uniformly distributed pseudorandom 2-dimensional input, ranging from 0 to 1.0.

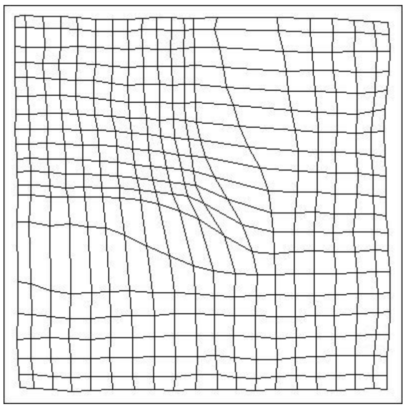

Figure 10. Same PLSOM as in figure 9 after 20000 further training steps with inputs ranging from 0 to 0.5. Note that while the weights have a higher density in the new input space, the same area as before is still covered, i.e. none of the old input space has been left uncovered.

3.2.3 Memory

If the opposite is the case, that 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. Figures 9 and 10 demonstrate what happens to a PLSOM if it is trained with pseudorandom, uniformly distributed 2-dimensional data in the $[0, 1]$ range for 50000 iterations and then presented with inputs confined to the $[0, 0.5]$ range for 20000 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.

3.3. 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 doesn't converge in the same manner as the SOM (there is always a small amount of movement).

4. Conclusion

The PLSOM 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 and preserves generality. This is achieved without inducing a significant computation time or memory overhead.

References

[1] C. M. Bishop, M. Svensén, and C. K. I. Williams. Gtm: A principled alternative to the self-organizing map. *Advances in Neural Information Processing Systems*, (9), 1997.

[2] C. M. Bishop, M. Svensén, and C. K. I. Williams. Gtm: The generative topographic mapping. *Neural Computation*, 10(1):215–235, 1998.

[3] B. Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, Cambridge MA, 1995.

[4] M. M. V. Hulle and K. U. Leuven. Globally-ordered topology-preserving maps achieved with a learning rule performing local weight updates only. *Neural Networks for Signal Processing [1995] V. Proceedings of the 1995 IEEE Workshop*, pages 95–104, Sep 1995.

[5] J. H. Kaas. Plasticity of sensory and motor maps in adult mammals. *Annual Review of Neuroscience*, 14:137–167, Mar 1991.

[6] T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, 1997.

[7] F. Mulier and V. Cherkassky. Learning rate schedules for self-organizing maps. In *Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on*, volume 2, pages 224–228. IEEE, 1994.

[8] D. R. Wilson and T. R. Martinez. The need for small learning rates on large problems. In *Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on*, volume 1, pages 115–119. IEEE, IEEE, July 2001.

[9] J. Zhu and E. Berglund. Jrobot. http://blog.no/erik.berglund/jrobot, 2002.