Cellular Self-Organising Maps - CSOM

Bernard Girau, Andres Upegui

Cellular Self-Organising Maps - CSOM

Bernard Girau[1] and Andres Upegui[2]

[1] Université de Lorraine, CNRS, LORIA, F-54000 Nancy, France,
bernard.girau@loria.fr,
[2] inIT, HEPIA, University of Applied Sciences of Western Switzerland, Switzerland
andres.upegui@hesge.ch

Abstract. This paper presents CSOM, a Cellular Self-Organising Map which performs weight update in a cellular manner. Instead of updating weights towards new input vectors, it uses a signal propagation originated from the best matching unit to every other neuron in the network. Interactions between neurons are thus local and distributed. In this paper we present performance results showing than CSOM can obtain faster and better quantisation than classical SOM when used on high-dimensional vectors. We also present an application on video compression based on vector quantisation, in which CSOM outperforms SOM.

Keywords: neural networks, self-organising maps, cellular computing

1 Introduction

The work presented in this paper is part of the SOMA project[3] (*Self-organising Machine Architecture* [1]). This project aims at developing an architecture based on brain-inspired self-organisation principles in a digital reconfigurable hardware. Self-organising neural models play a central role in this project. As we wish to deploy the architecture in a manycore substrate, the scalability and decentralisation of the architecture needs to be ensured.

Cellular computing approaches [2] appear as a promising solution for tackling the scalability limitations of current hardware implementations of self-organising maps. Cellular computing refers to biological cells that interact with their neighbour cells, but not with remote ones, at least not directly. While parallel computing deals with a small number (tens up to tens of thousands in supercomputers) of powerful processors able to perform a single complex task in a sequential manner, cellular computing is based on another philosophy: simplicity of basic processing cells, their vast parallelism, and their locality. The two latter properties induce another fundamental property: decentralisation. In the SOMA project, we propose to combine cellular and neural properties to enable cellular structures to behave like biologically-inspired neural models. In this context, we have defined a cellular version of self-organising maps, that we call CSOM. The aim

[3] The authors thank the Swiss National Science Foundation (SNSF) and the French National Research Agency (ANR) for funding the SOMA project ANR-17-CE24-0036.

of this paper is to study the specific properties of this model in terms of vector quantisation performance. We do not focus on scalability and decentralisation aspects, though such requirements are our first motivation to define CSOM.

This paper describes CSOM in section 2 by comparing it to standard SOM. We present the algorithm in detail and we explain its behaviour. Then in section 3, we present two experiments. The first aims at comparing and analysing the behaviour of CSOM and SOM when learning increasingly dimensional and sparse data. The second experiment compares SOM and CSOM algorithms for video compression by replacing a sequence of video frames (images) by a codebook of thumbnails learnt by SOM and CSOM. Section 4 concludes with perspectives on other aspects of the SOMA project.

2 Self-organising maps : SOM and Cellular SOM

2.1 SOM: self-organising maps

Self-organising maps (SOM), initially proposed by Kohonen [3], consist of neurons organised on low-dimensional arrays (most often 2D) that project patterns of arbitrary dimensionality onto a lower dimensional array of neurons. Each neuron has an associated weight vector, or prototype vector. Its dimension is defined by the nature of input data, not by the dimensionality of the neural array. All neurons receive the same input pattern and an iterative mechanism updates the neuron's weights so as to learn to quantise the input space in an unsupervised way. This mechanism first selects the neuron whose weights best fit the given input pattern, and then brings the weights of all neurons more or less closer to the current input pattern, depending on the distance to the winner neuron in the neural array (Fig. 1). As with any vector quantisation method, the result is a codebook: it is the set of all neuron weight vectors, or codewords. Self-organising maps are known to define "topological" codebooks: after learning, two vectors that are close in the input space will be represented by codewords of close neurons in the neural map. Thus, neighbouring neurons in the map have similar weight vectors.

2.2 CSOM: cellular self-organising maps

As Kohonen's SOM, CSOM is a vector quantisation algorithm which aims to represent a probability density function into a codebook, i.e. a set of prototype vectors or codewords. Is is also a neural network composed of an n-dimensional array of neurons. In this paper we will only consider the case of the 2-dimensional architecture. The main difference with SOM is that the neighbourhood notion is replaced by a notion of local connectivity and propagation of influence. Each neuron has a number of associated synapses that define which neuron will have an influence onto which other. Synapses can be seen as interconnection matrices. In this paper, we assume that synapses are simply interconnecting every neuron to its four physical neighbours. With this restriction, the neural structure of CSOM is similar to the 2D grid of the usual SOM. But more complex synaptic interconnections can be used, such as resulting from pruning as in [4].

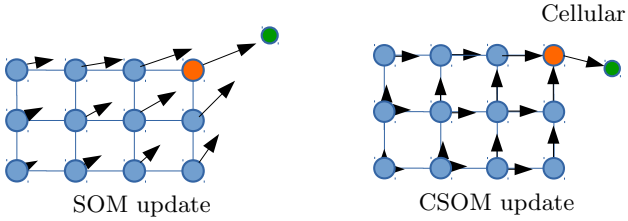


Fig. 1. Weight update rule on SOM and CSOM.

Unlike SOM, CSOM performs a weight update rule which moves network weights towards neighbour neurons weights instead of input patterns. Figure 1 illustrates this update: the green node represents a new input pattern, the orange node represent the BMU, which updates its weight towards the new pattern in both cases, and the blue nodes represent the remaining neurons on the SOM. While in SOM, weight update is influenced by every new input pattern in a direct manner, in CSOM the influence is indirect: neuron weights are influenced by the neighbour weights, starting from the BMU.

2.3 Algorithms

Common notations and mechanisms Each neuron in a SOM or CSOM is represented by a d-dimensional weight vector, $\mathbf{m} \in \mathbb{R}^d$, also known as prototype vector or codeword, $\mathbf{m} = [m_1, ..., m_d]$, where d is the dimension of the input vectors, $\mathbf{x}$. Neurons are located in a n-dimensional map (Usually $n = 2$). In this map, $\mathbf{r}_i$ are the coordinates of neuron i and $\mathbf{m}_i$ is its weight vector. For SOM, these coordinates are the basis of distance computations to define neighbourhood relations. For CSOM, we simply consider in this paper that synaptic interconnections are set to connect all adjacent neurons in the map.

In the algorithm of both SOM and CSOM, learning starts with an appropriate (usually random) initialisation of the weight vectors, $\mathbf{m}_i$. Input vectors are presented to the neural map. For each input vector $\mathbf{x}$, the distance from $\mathbf{x}$ to all the weight vectors is calculated using a distance measure noted $\|\cdot\|$, which is typically the Euclidean distance. The neuron whose weight vector gives the smallest distance to the input vector $\mathbf{x}$ is called the best matching unit (BMU), denoted by c, and determined according to: $\|\mathbf{x} - \mathbf{m}_c\| = \min_i \|\mathbf{x} - \mathbf{m}_i\|$

SOM algorithm The basic version of SOM learning is an on-line stochastic process[4] which has been inspired by neurobiological learning paradigms, but other extensions exist [5, 6]. For each iteration, an input is randomly drawn, and its BMU c is computed. Then all neurons i update their weights according to:

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + \epsilon(t)(\mathbf{x}(t) - \mathbf{m}_i(t))e^{-\frac{\|\mathbf{r}_c - \mathbf{r}_i\|^2}{2\sigma^2(t)}} \quad (1)$$

where t denotes the time/iteration, $\mathbf{x}(t)$ is an input vector randomly drawn from the input data set at time t, and $\epsilon(t)$ the learning rate at time t. The

[4] A batch version of SOM learning exists, but the on-line mode training provides us with a more detailed evolution to observe, and we apply our models to applications with huge redundant learning databases for which batch-learning is quite inefficient.

learning rate $\epsilon(t)$ defines the strength of the adaptation, which is application-dependent. Commonly $\epsilon(t)$ is constant or a decreasing scalar function of t. The term $\|\mathbf{r}_c - \mathbf{r}_i\|$ is the distance between neuron i and the winner neuron c, and $\sigma(t)$ is the standard deviation of the gaussian neighbouring kernel.

CSOM algorithm The CSOM learning algorithm is an on-line mode training. For each iteration, an input is randomly drawn, and its BMU c is computed. First the BMU updates its weight vector according to: $\mathbf{m}_c = \mathbf{m}_c + \alpha(t)(\mathbf{x}(t) - \mathbf{m}_c)$ where $\alpha(t)$ is the learning rate at time/iteration t. Then every other neuron $i \neq c$ in the map updates its weight vector as follows:

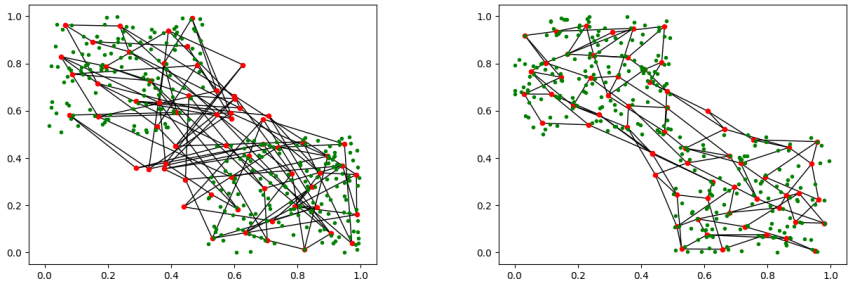
$$\mathbf{m}_i = \mathbf{m}_i + \alpha(t)\frac{1}{\#Infl(i)} \sum_{j \in Infl(i)} (\mathbf{m}_j - \mathbf{m}_i) e^{\left(-\frac{\sqrt{d}}{\eta(t)} \frac{hops(c,i)}{||\mathbf{m}_i - \mathbf{m}_j||}\right)} \qquad (2)$$

where $hops(c, i)$ is the number of propagation hops to reach neuron i from the BMU through synaptic connections, $Infl(i)$ is the set of the influential neurons of neuron i, and $\#Infl(i)$ is the number of influential neurons for neuron i. An influential neuron of neuron i is defined as the neuron from which neuron i received the propagating learning signal. For a neuron with $hops(c, i) = h$, the influential neuron(s) will be every neuron j connected to i with $hops(c, j) = h-1$. The parameter $\alpha(t)$ is the learning rate at time t, and $\eta(t)$ is the elasticity of the network at time t. This latter parameter is modulated by $\sqrt{d}$ so as to take into account the range $[0, \sqrt{d}]$ of euclidean distances in dimension d.

The principle of this algorithm results from a combination of cellular computing principles and dynamic adaptation as found in the Dynamic SOM (DSOM) model [7]. Each neuron weight vector gets influenced by the weight vectors of its influential neurons through which the cellular propagation signal has been received, instead of being influenced by the input vector through its BMU. The learning rate is modulated by the proximity of the weight vector with the weight vectors of its influential vectors, so that learning gets faster when close neurons do not have close weight vectors. The BMU first influences lateral neurons directly connected to it (for them $hops = 1$) through synapses. Equation 2 updates the weight of neighbouring neurons by attracting their weight towards the winning codeword $\mathbf{m}_c$. After being updated, these neighbouring neurons will update their own neighbouring neurons by means of the same equation 2. This results in a gradient permitting the overall network weights to get influenced by every new input vector through local influences. The elasticity parameter can be understood as how much the resulting map is expected to maintain its underlying structure in the codeword space. In figure 2, for instance, a lower elasticity results in a final network topology closer to its initial random configuration.

Complexity From a sequential[5] complexity point of view, SOM and CSOM have a comparable cost. If we consider that exponential values are pre-computed and stored (neural distances as well as numbers of *hops* can only take a finite number of values), the main computational difference between them is that all

[5] The question of a parallel complexity comparison between SOM and CSOM is not discussed here, even if the cellular version targets parallel hardware implementations.



(a) Elasticity : $\eta(0) = 2$ (b) Elasticity : $\eta(0) = 20$

Fig. 2. Learned codebook for CSOM with a 2D distribution with two main clusters, using different elasticity parameters

influential neurons are taken into account in CSOM, thus inducing an approximately doubled complexity for CSOM with respect to SOM since most neurons have two influential neurons in a 2D structure. Let us mention the fact that randomly choosing only one influential neuron for each neuron leads to similar behaviors and leads to the same computational complexity for the two models[6].

Parameter tuning SOM and CSOM algorithms depend on the number of learning iterations I. They also depend on parameters that evolve along training: $\epsilon(t)$ and $\sigma(t)$ for SOM, $\alpha(t)$ and $\eta(t)$ for CSOM. Classically, these parameters linearly decrease during training from an initial value to a final value. Thus, a fair comparison of SOM and CSOM raises the question of choosing ϵ_{init}, ϵ_{final}, σ_{init}, σ_{final}, α_{init}, α_{final}, η_{init}, η_{final}, I_{SOM} and I_{CSOM}. In this paper, we set these parameters by means of a simple "manual" exploration for our tests on artificial distributions, and by means of an optimisation by genetic algorithm in the case of a video compression application. In these two cases, using the same optimisation technique for SOM and CSOM thus provides performance comparisons on a fair basis even if the choice of each optimisation technique rather than another one can be discussed.

3 Experimental setup and Results

We propose here two kinds of tests: learning more or less sparse data generated from artificial statistical distributions in variable dimensions, then learning to quantise images in a video compression application.

3.1 Quantisation of artificial *d*-dimensional distributions

We have tested our algorithm by presenting input vectors drawn from statistical distributions in various dimensions. In dimension d, the chosen distribution uses

[6] It is important to notice that this complexity equivalence requires anyway a careful handling of the iteration order among neurons of CSOM: influential neurons must be updated before the neuron they influence. A simple possible way to ensure this property is to consider that the BMU position defines four sectors (top-left, top-right, bottom-left, bottom-right) in the neural map and to update weights sector per sector, always starting from the BMU corner.

d uniform clusters of data with a δ sparsity coefficient: data are generated as $\{x_1 = U([0, 1-\delta]), \ldots, x_{j-1} = U([0, 1-\delta]), x_j = U([\delta, 1]), x_{j+1} = U([0, 1-\delta]), \ldots, x_d = U([0, 1-\delta])\}$, where the coordinate number j is randomly chosen for each new generated vector, and $U([a, b])$ is the uniform distribution in $[a, b]$. When $\delta = 0$, it results in d-dimensional uniform distributions. When $\delta = 0.8$ data are distributed in d rather small clusters for which all coordinates are between 0 and 0.2, except one that is between 0.8 and 1. These distributions are illustrated in 3D on figure 3.

In the reported experiments, the network topology has been defined as a 2-D array mesh of 8x8 neurons. Each neuron has 4 synapses connected to neighbour neurons. Weights have been uniformly initialised with $U([0, 1])$. Parameters ϵ_{init}, ϵ_{final}, σ_{init}, σ_{final}, α_{init}, α_{final}, η_{init}, η_{final}, I_{SOM} and I_{CSOM} have been optimised by means of a simple "manual" exploration method, considering the small computation time. The performance is measured by the average quantisation error (AQE) computed on independent test sets drawn from the same distributions:

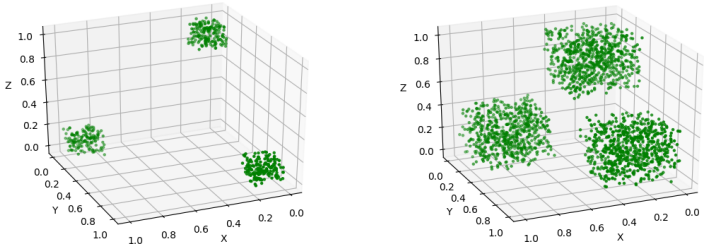
$$AQE = \frac{1}{K} \sum_{k=1}^{K} ||\mathbf{x}_k - \mathbf{m}_{BMU(\mathbf{x}_k)}||^2$$

where K is a number of input test vectors, and $\mathbf{x}_k$ is one of them. In the experiments presented here, K has been set to 500, while each learning epoch uses 100 input vectors, randomly drawn from the distribution at each learning iteration.

Figure 4 shows the ratio of the results $\frac{AQE_{CSOM}}{AQE_{SOM}}$ obtained for 50 runs on several dimensions and for sparsity coefficients $\delta = 0$ and $\delta = 0.9$. An increase in dimensions favours CSOM, especially for sparse distributions. The main property of CSOM is a faster convergence in most cases. We assume that the decorrelation between elasticity and weight attraction in CSOM makes it able to tolerate less organised codewords, at least temporary, so that this model does not require to attract all neuron weights close to each other at the beginning of learning before unfolding in the input space, as it is usually observed with SOM. Nevertheless we have not yet been able to quantify this phenomenon and further studies are still required to validate this assumption.

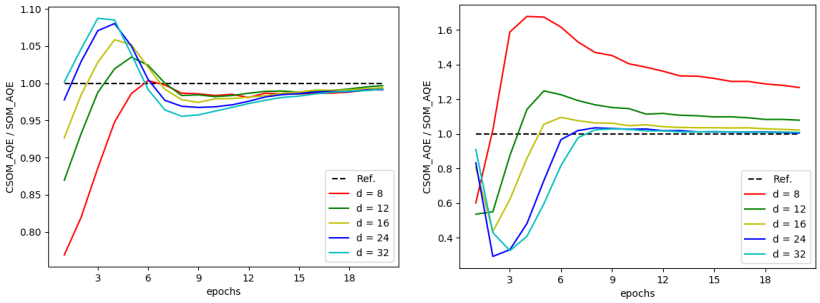
3.2 Video compression

In order to evaluate how CSOM behave in real-world applications, we have decided to train them on data extracted from image sequences. Self-organising



(a) Sparsity $\delta = 0.8$ (b) Sparsity $\delta = 0.55$

Fig. 3. Statistical distribution in 3D for $\delta = 0.8$ and $\delta = 0.55$



(a) Sparsity $\delta = 0$ (b) Sparsity $\delta = 0.9$

Fig. 4. AQE ratio $\frac{AQE_{CSOM}}{AQE_{SOM}}$ evolution with various dimensions

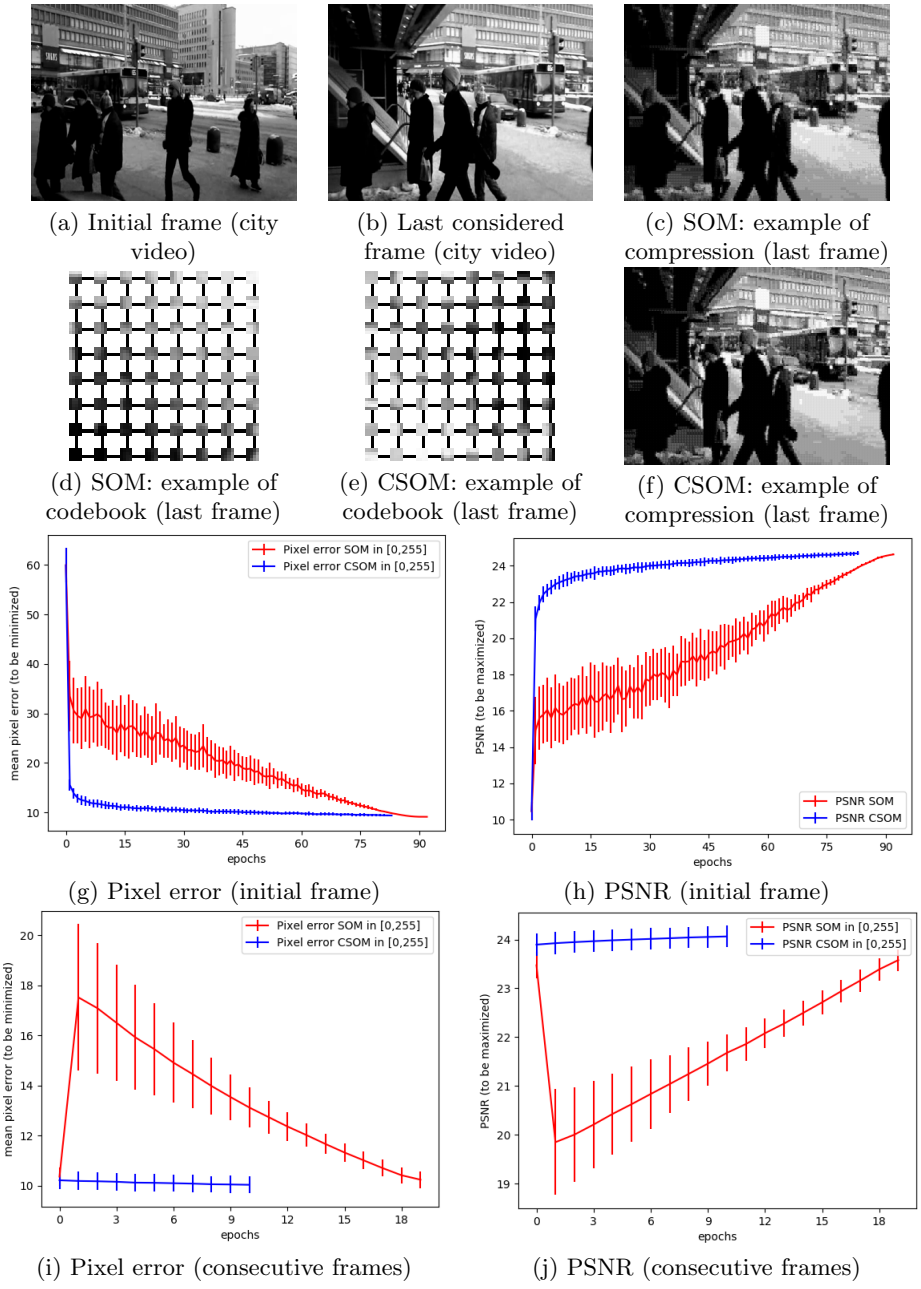
maps can be interestingly applied to lossy image compression [8–10, 4]. The principle is to split the image into non overlapping thumbnails, then learn a good quantisation of these thumbnails. Compressing the image is performed by replacing each thumbnail by the index of the closest codeword (or codeword of its BMU). The list of thumbnail indexes is further compressed by classical methods such as entropy coding. Uncompressing the image can be performed by replacing each decoded index by the corresponding thumbnail (codeword). The result is similar to the original image, but with every thumbnail replaced by the codeword learned by its BMU (see further an example on figure 5(c)). Compared to other quantisation techniques, SOM preserve topological properties, so that close parts of an image that are often similar will be coded by neighbouring neurons. It makes it possible to further reduce the size of the compressed file by efficiently coding codeword indexes thanks to a differential coding performed before entropy coding [9].

In order to see how fast CSOM and SOM respectively adapt to changing data, we extend this application to video compression as follows:

1. The first $L \times H$ frame is split into thumbnails of $l \times h$ pixels.
2. A SOM and a CSOM of $n \times n$ neurons learn these thumbnails, using a set of parameters optimised for processing a first frame (see below). Each neuron has a $l \times h$ weight vector, that can be interpreted as a thumbnail codeword.
3. Each consecutive frame is processed in the following way:
   (a) The $L \times H$ frame is split into thumbnails of $l \times h$ pixels.
   (b) The previously learned SOM and CSOM learn these new thumbnails, using a set of parameters optimised for processing consecutive frames.

As an example, we consider sequences of 384×288 images subdivided into 6912 thumbnails of 4×4 pixels. Using a 8×8 SOM or CSOM ($b = 6$), the compression ratio already reaches a very high value of 7 before even further compressing the index list using differential and entropy coding (see [10] for details on computing the exact compression ratio).

For each model and image sequence, we optimise the main parameters of the learning algorithms (ϵ_{init}, ϵ_{final}, σ_{init}, σ_{final}, α_{init}, α_{final}, η_{init}, η_{final}, I_{SOM} and I_{CSOM}) by means of a genetic algorithm with 0.1 mutation probability, 20 % of the population considered as elite, 32 individuals in the population and



(a) Initial frame (city video)

(b) Last considered frame (city video)

(c) SOM: example of compression (last frame)

(d) SOM: example of codebook (last frame)

(e) CSOM: example of codebook (last frame)

(f) CSOM: example of compression (last frame)

(g) Pixel error (initial frame)

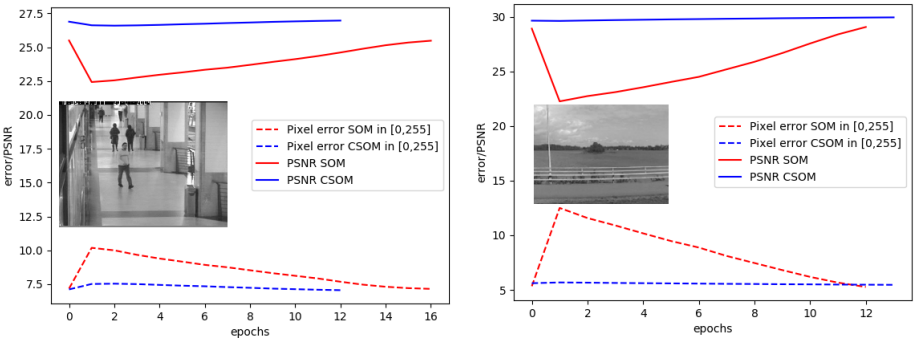
(h) PSNR (initial frame)

(i) Pixel error (consecutive frames)

(j) PSNR (consecutive frames)

Fig. 5. SOM (red) vs CSOM (blue) for lossy video compression (city video example)

20 generations, each individual fitness being evaluated by learning 20 randomly initialised SOM or CSOM. We compute the performance of the resulting SOM and CSOM after each "epoch" of 216 randomly chosen thumbnails among the 6912 available. Two sets of parameters are evolved for each model and image



(a) Mean pixel error and PSNR evolution (consecutive frames of corridor video)

(b) Mean pixel error and PSNR evolution (consecutive frames of outdoor video)

Fig. 6. SOM (red) vs CSOM (blue) for corridor and outdoor video examples

sequence: one for learning the quantisation of the first frame, the other one for learning the quantisation of a frame after having learned the quantisation of the previous frame in the sequence. In this second case, the number of learning iterations per "epoch" is reduced to 43, since learning the next frame is quite fast when the SOM or CSOM has already learned the previous frame so that its codewords are already very satisfactory with respect to the new frame.

Based on experiments using various image sequences from the CAVIAR project [11] as well as from the CVD project [12], CSOM significantly improve the mean pixel error (normalised in [0,255]) with respect to standard SOM. We also observe a significant perceptual improvement of the visual result that is illustrated by the usual peak signal-to-noise ratio (PSNR) to be maximised:

$$\text{PSNR} = 10 \times \log_{10}(\frac{255^2}{\text{MSE}}) \quad \text{where} \quad \text{MSE} = \frac{1}{L \times H} \sum_{x=1}^{L} \sum_{y=1}^{H} (P_{x,y} - P'_{x,y})^2$$

Figure 5 shows results of SOM and CSOM learning to compress a video taken in a city with many buildings and moving pedestrians in addition to a circular movement of the camera (see 5(a) and 5(b)). Learning results for the first frame are averaged on 32 randomly initialised maps. CSOM main advantages are a faster and smoother learning (see 5(g) for mean pixel error and 5(h) for PSNR). CSOM learning of consecutive frames also takes advantage of a more progressive performance when dealing with the next 30 frames of this sequence, always using the same evolving 32 SOM and CSOM: while SOM first seem to unlearn their previous satisfactory codebook before converging again towards an even more satisfactory one for the newly presented frame, CSOM do not suffer such a temporary degradation of their performance (5(i) for mean pixel error and 5(j) for PSNR). Similar results are obtained for other videos though sometimes less contrasted (see for example figure 6(a) with a stationary camera in a commercial centre), or even in the few cases when SOM reach a slightly better performance for the first frame (see for example figure 6(b) with a shaky camera in a circular movement outdoor).

4 Conclusion

This paper presented the Cellular SOM algorithm, a Self-Organising Map based on cellular computing. The behaviour exhibited by the algorithm on the experiments presented in this paper shows that CSOM outperforms classical SOM when applied to high dimensional sparse data. It can achieve a better quantisation and is faster than SOM. However, on lower dimensional and less sparse data SOM can be better. The fast algorithm convergence of CSOM is of particular interest for dynamic problems. When data statistical properties change over time, at it can be the case on a video, CSOM is able to re-adapt faster in order to match new input vectors, maintaining a low quantisation error over time.

Further work will focus on including pruning and sprouting mechanisms [4] in order to remove and re-create synapses or create connections with remote neurons. This feature should lead to a better quantisation of sparse data, and rebuild the network when the dynamicity of the problem may impose it. CSOM has been initially designed taking into account the constraints imposed by cellular computing: a cellular substratum of processing elements. Future works will implement CSOM on a real cellular hardware architecture in order to drive the self-organisation of a manycore hardware according to new input data, as targeted by the SOMA project.

References

1. L. Khacef, B. Girau, N. P. Rougier, A. Upegui, and B. Miramond, "Neuromorphic hardware as a self-organizing computing system," in *NHPU : Neuromorphic Hardware In Practice and Use*, IEEE, Ed., Jul. 2018, pp. 1–4.
2. M. Sipper, "The emergence of cellular computing," *Computer*, vol. 32, no. 7, pp. 18–26, 1999.
3. T. Kohonen, "The self-organizing map," vol. 78, no. 9, 1990, pp. 1464–1480.
4. A. Upegui, F. Vannel, B. Girau, N. Rougier, and B. Miramond, "Pruning self-organizing maps for cellular hardware architectures," in *NASA/ESA conf. on Adaptive Hardware and Systems*, 2018.
5. M. Cottrell, M. Olteanu, F. Rossi, and N. Villa-Vialaneix, "Theoretical and applied aspects of the self-organizing maps," in *Advances in self-organizing maps and learning vector quantization*. Springer, 2016, pp. 3–26.
6. T. Kohonen, "Essentials of the self-organizing map," *Neural Networks*, vol. 37, pp. 52–65, Jan. 2013.
7. N. P. Rougier and Y. Boniface, "Dynamic Self-Organising Map," *Neurocomputing*, vol. 74, no. 11, pp. 1840–1847, 2011.
8. Z. Huang, X. Zhang, L. Chen, Y. Zhu, F. An, H. Wang, and S. Feng, "A hardware-efficient vector quantizer based on self-organizing map for high-speed image compression," *Applied Sciences*, vol. 7, no. 11, p. 1106, 2017.
9. C. Amerijckx, J.-D. Legat, and M. Verleysen, "Image compression using self-organizing maps," *Systems Analysis Modelling Simulation*, vol. 43, no. 11, pp. 1529–1543, 2003.
10. Y. Bernard, E. Buoy, A. Fois, and B. Girau, "NP-SOM: network programmable self-organizing maps," in *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2018, pp. 908–915.
11. "EC funded CAVIAR project," http://homepages.inf.ed.ac.uk/rbf/CAVIAR/.
12. "CVD video database," http://www.helsinki.fi/~tiovirta/Resources/CVD2014/.