Self-Organizing Maps and DSOM

From unsupervised clustering algorithms to models of cortical plasticity

Research Project Report – "Modeling in Neuroscience" course

Lilian Besson*
Department of Mathematics
École Normale Supérieure de Cachan (France)
lilian.besson@ens-cachan.fr

Abstract

For this mini-project, I studied computational models for unsupervised learning, starting from a classical one (K-Means), and then some models inspired by neuroscience or biology: Neural Gas, Neural Fields, and Self-Organizing Maps (SOM). I mainly focused on the later, and on a variation of the SOM algorithm where the original time-dependent (learning rate and neighborhood) learning function is replaced by a time-invariant one, giving the Dynamic SOM (DSOM).

This allows for on-line and continuous learning on both static and dynamic data distributions. One of the property of the newly proposed algorithm is that it does not fit the magnification law and the achieved vector density is not directly proportional to the density of the distribution as found in most vector quantization algorithms. It also has the advantage of requiring only two parameters and not five, making it easier to automatically tune them with a manual exploration or a grid search.

From a neuroscience point of view, this dynamic extension of the SOM algorithm sheds light on cortical plasticity seen as a dynamic and tight coupling between the environment and the model. The difference between SOM and DSOM can be seen as the natural difference between early years learning (in child) and long-term learning (in adults).

*If needed, see on-line at http://lbo.k.vu/neuro2016 for an e-version of this report, as well as additional resources (slides, code, figures, complete bibliography etc), open-sourced under the MIT License.

Contents

Project Advisor: Nicolas P. Rougier (MNEMOSYNE team, Inria Bordeaux)
Course: "*Modeling in Neuroscience*", by J.-P. Nadal, in 2016,
Master 2 **program:** Mathématiques, Vision, Apprentissage (MVA)
at ENS de Cachan.
Grade: I got 17.5/20 for my project.

Outline: In this report, we start by introducing the problem of unsupervised learning, and the main approach we studied, in section 1. Then different classical models are presented in section 3 (Neural Gas, Neural Fields and Self-Organizing Maps), and each comes with a short discussion about its parameters; followed in section 4 by the presentation of a new unsupervised clustering algorithm: Dynamic Self-Organizing Maps [RB11a]. This model is a simple relaxation of the well-known SOM algorithm, where the learning rate and neighborhood width both become time-invariant but local-dependent. We will also detail some experiments, comparing NG, SOM and DSOM on several types of both static and dynamic distributions; and exploring the effects of the 2 parameters of a DSOM model. The DSOM is also applied to a highly-non-stationary distribution, proving it can be a satisfying model of long-term learning as encountered in the cortex ("cortical plasticity"). At the end, we conclude with a short sum-up in section 5, along with a list of references, and links to additional on-line resources in appendix A.
Note: This report comes along the slides used for the oral presentation, which covers a similar work, but you might also be interested in reading them[0].

---

1 Introduction

Quick overview of the goals of this project

In machine learning, and in the brain [Doy00], there is different kinds of learning: Supervised learning (as found in the cerebellum), Reinforcement learning (as found in the basal ganglia and the thalamus), and **Unsupervised learning** (as found in the cortex).

A lot of unsupervised learning models exist, and we will first focus on K-Means, a very classical one, and then on some models inspired from neuroscience: Neural Gas, Neural Field & Dynamic NF are presented very quickly, and then **Self-Organizing Maps** & **Dynamic SOM** are studied more in details.

Unsupervised learning has many applications, including data/image compression (e.g. color quantization, as used by the GIF image format), automatic clustering, visualization, etc; and modeling self-organization and online learning (plasticity) in the cortex is only one of the possible application of these algorithms.

---

2 Unsupervised Learning, starting with K-Means

2.1 Different types of learning

In Machine Learning: Each type of learning have been thoroughly studied from the 50's:

- **Supervised** (or deep) learning means learning from labeled data; for a reference see e.g. [Bis06]. A very successful application of deep learning from a tremendous quantity of data is the Google Images application (`images.google.com`), which showed in 2012 that images retrieval works in the real-world (almost any image, from anywhere around the globe, from movies, from comic books etc, is recognized very quickly).

[0] You can find them online: `https://goo.gl/GjrwkX` or here.

– **Reinforcement** learning means learning with feedback (also referred to as reward or penalty); for a reference see e.g. [SB98]. Another example of a successful application of reinforcement learning is the very recent success of Google DeepMind's Alpha Go project, which showed that reinforcement learning (and deep learning) can give very powerful AIs (the first AI to ever beat a professional Go player).

– *But* **unsupervised learning** is still **the harder**, the "Holy Grail" of machine learning.

As a lot of studies have shown, the main three different types of learning can be found in the brain [RB11a, Doy00]. We will not give more details here neither about the different types learning nor the neuroscience background and experiments that permitted to the neuro-biology community to agree on this representation (Fig. 1):

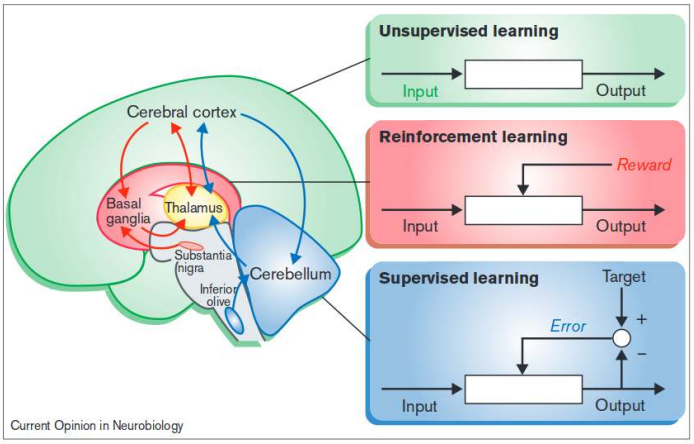


Figure 1: The 3 main types of learning are present in the brain [Doy00, Figure 1].

Why is *unsupervised* learning harder? In the unsupervised setting, the algorithm has absolutely no idea what the data is: there is no labels, no time organization, and no feedback/reward/penalty: *just* ***raw*** *data*.

As many specialist accord to say, predictive learning is **the future**. A very recent quote from Richard Sutton[1] and Yann LeCun[2] illustrates this:

> "AlphaGo is missing one key thing: the ability to learn *how the world works*." Richard has long advocated that the ability to predict is an essential component of intelligence. **Predictive (unsupervised) learning** is one of the things some of us see as the next obstacle to better AI.

Figure 2: Yann LeCun quoting Richard Sutton in February 2016.

2.2 Vectorial quantization: a simple *unsupervised* task

Let $X = \{\mathbf{x}_1, \dots, \mathbf{x}_p\}$ be samples in a space E. We want to cluster the data, and this directly raises a few questions:

– How to cluster similar data together? Similar in what sense?

[1] One of the father of reinforcement learning, cf. [SB98].
[2] One of the father of deep learning.

– How many groups there is? K *clusters* C_j: find K.
– What are the best representatives of each group? *"Centroids"* μ_j.
– Can we identify close groups (and merge them) ?

For instance, in Fig. 3 below is showed points in $2D$ (i.e. $X \subset \mathbb{R}^2$), and our human eyes directly see they are organized in groups ("clusters"), about 16 of them. A first clustering, easy to obtain, is shown in the middle, it consists in dividing the square in 16 small sub-squares, all equal (their centers is shown as a big white circle). A second clustering, way more visually satisfying, is shown on the right: this one consists of 16 Voronoï diagrams, and is obtained with a Delaunay algorithm.

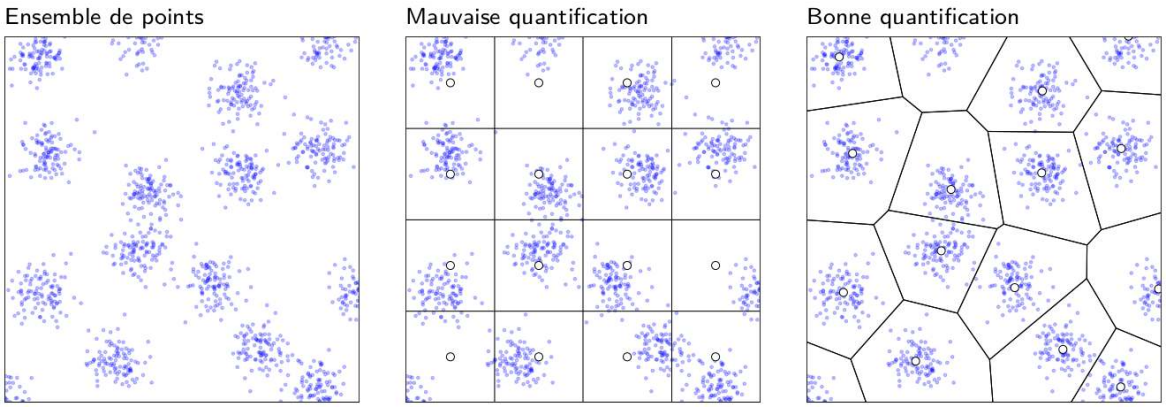


Figure 3: For 2D points, examples of a bad quantization and a good quantization

Let start by clarifying mathematically what a **vectorial quantization** algorithm can be. Let E be the data space (containing the dataset $X \subset E$), a compact manifold[3] in $\mathbb{R}^d$.

Definition 2.1. *A **vectorial quantization** of E is simply defined by a function Φ, and a set $Q \subset E$, so that $\forall \mathbf{x} \in E, \Phi(\mathbf{x}) \in Q$ (i.e. $\Phi : E \to Q$).*
*Q is usually finite, or at least discrete, and is called the **codebook**: $Q = \{w_1, \dots, w_n\}$.*

Examples 2.2. *Assume we have data lying in $E = \mathbb{R}$ (real line), and we want to quantize them.*
First, for a discrete and finite *codebook, $Q = \{\pm 1\}$: we can take $\Phi(x) = \operatorname{sign}(x)$, there is only 2 prototypes $(w_i)_{i=1..n}$.*
Secondly, for a discrete but infinite *codebook, $Q = \mathbb{Z}$: we can take $\Phi(x) = \lfloor x \rfloor$ or $\lceil x \rceil$, here there is an infinite number of prototypes $(w_i)_{i=1..n}$.*

So a natural question is: can we **generalize** to any data? We would like to automatically find the target/compressed set Q (the codebook), and the clustering function Φ, for *any dataset* X in a set E ?

We can sum up the notations used hereafter, and express mathematically the goal. For a finite codebook $Q = \{w_1, \dots, w_n\}$, we define the clusters by $C_i \overset{\text{def}}{=} \{\mathbf{x} \in E : \Phi(\mathbf{x}) = \mathbf{w}_i\}$. We assume the data $\mathbf{x}_j$ are drawn from a target probability density f on E.

First, define the (continuous) *distortion* of the VQ as:

$$\mathbf{J}(\Phi) \overset{\text{def}}{=} \sum_{i=1..n} \mathbb{E}_{f,C_i}\Big[\|\mathbf{x} - \mathbf{w}_i\|^2\Big] = \sum_{i=1..n} \int_{C_i} \|\mathbf{x} - \mathbf{w}_i\|^2 \, f(\mathbf{x}) \, \mathrm{d}\mathbf{x}. \tag{1}$$

[3] As no mathematical proofs of convergence, correctness or stability are done in the report, this hypothesis is not used. But it is a classical assumption, usually required for the few theoretical work on the SOM and DSOM algorithms, see for instance [CFP87, CFP98].

But as always in machine learning and inference, the target distribution f *is unknown*. We assume that only unbiased observations $\mathbf{x}_j$ are available ($j = 1..p$), and so we define the *empirical distortion* as $\hat{\mathbf{J}}(\Phi) \overset{\text{def}}{=} \frac{1}{p} \sum_{i=1}^{n} \sum_{\mathbf{x}_j \in C_i} \|\mathbf{x}_j - \mathbf{w}_i\|^2$. And the goal is to *minimize the empirical distortion* $\hat{\mathbf{J}}$.

A "classical" problem The vector quantization problem, as defined above, is nothing but a clustering problem, and this has been studied a lot in the last 30 years.
As a consequence, plenty of algorithms have been proposed, including K-Means (1), Elastic Net (a L1-L2 penalized least-squares regression), **(Dynamic) Self-Organizing Map** (2) [RB11a], and (Growing/Dynamic) Neural Gas (3), (Dynamic) Neural Field (4) [RD11].
And vector quantization counts many applications, like compression of data (images etc), automatic classification/categorization[4] etc.

Additionally to these applications, an interesting consequence of the NG, SOM and DSOM models will be there connexions with the learning processes as found in human or primates (see later).

2.3 K-Means: a first unsupervised algorithm

Before studying clustering models inspired from biology, let start by quickly reviewing a well-known one: the K-Means algorithm.

It clusters data by trying to separate the p samples $\mathbf{x}_i$ in K groups of equal variance, minimizing a criterion known as the "*distortion*" $\mathbf{J}(\Phi)$. This algorithm requires K, the number of clusters, to be specified before-hand, as most unsupervised models. There is strategies to try to find a good value for K automatically (based on a grid-search), but the question of the "best possible K" is mathematically unfunded.

K-Means also has the advantage of scaling well to large number of samples, and as a consequence it has been used across a large range of application areas in many different fields. For example, Fig. 4 below shows 10 clusters, obtained by K-Means from the well known MNIST hand-written digits dataset[5], after a dimension reduction of the digit dataset to 2 dimensions.

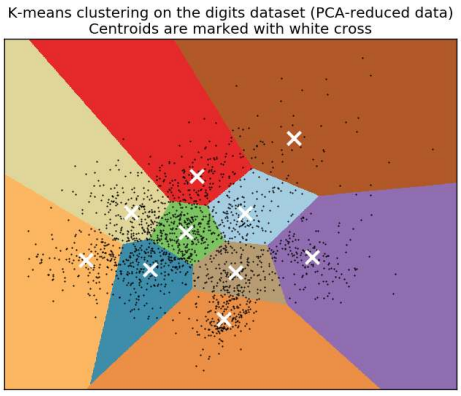


Figure 4: Example: K-Means clustering on the digits dataset (PCA-reduced data).

[4] E.g. in 2013 Netflix "automatically" discovered the main movie genres from its database of movies ratings.
[5] In `scikit-learn` [PVG+11], it is obtained with `datasets.load_digits`.

Description of the K-Means algorithm K-Means aims at dividing a set of p samples $X = \{\mathbf{x}_1, \dots, \mathbf{x}_p\}$, into K disjoint clusters C_j, each described by the *mean* μ_j of the samples in the cluster. The means are called the cluster "centroids"[6]. Aims to choose centroids that minimize the *distortion*:

$$\mathbf{J}(\Phi) = \frac{1}{p} \sum_{i=1}^{p} \min_{\mu_j \in C} \left(\|\mathbf{x}_i - \mu_j\|^2 \right). \tag{2}$$

Convergence & implementation K-Means is equivalent to the Expectation-Maximization algorithm with a small, all-equal, diagonal covariance matrix. And the E-M algorithm converges, as it strictly minimizes the distortion at each step. So K-Means converges indeed, but it can fall down to a *local minimum*: and that is one of the reason why a *dynamic* unsupervised learning algorithm can be useful.

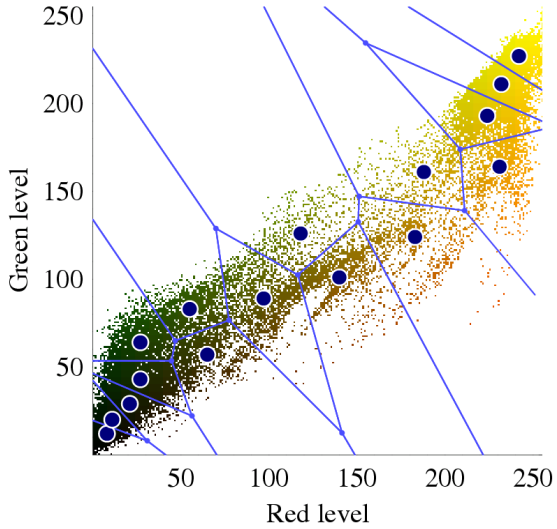
K-Means is quick and efficient (with K-Means++ initialization), usually converges well, and is easy to implement. It is available in `scikit-learn` [PVG+11], as `clustering.KMeans`, and for this project I also reimplemented it myself, see `kmeans.py` (on-line).

2.4 Application: color quantization

A nice application of any clustering algorithm can be color quantization for pictures or movies [Blo08].

Let start with a simple example, on a picture of a flower, with only two color channels (green and red), in order to visualize easily the color space as a $2D$ space (red/green). In this color-space, we can cluster all its colors (being the data $\mathbf{x}_j$), into only 16 **Voronoï diagrams** (16 is an arbitrary choice), as shown in Fig. 5b below.

(a) Flower "Rosa gold glow" (from Wikipedia).



(b) In the red/green color space.

Figure 5: Color quantization on a two-color flower picture (red/green).

An important observation to make from this first experiment is the well-known fact, that K-Means fits what is often referred as the "**magnification law**":

High density regions tend to have more associated prototypes than low-density regions.

[6] Note that they are **not**, in general, points from X (although they live in the same space).

Color quantization for a real-world photo Let apply[7] a color quantization compression on a larger photo[8], of 3648×2736 pixels, and 75986 colors.

(a) With a random codebook. (b) With a *K-Means codebook* (optimal).

Figure 6: Color quantization, from 75986 to 32 colors.

We clearly see that the compressed picture on the right (compressed with a codebook obtained by K-Means clustering) is visually better (and it is clearer on the HD pictures). They offer a (theoretical) compression factor of $75986/32 = 2374 \simeq 2300$: that's **huge**!

In practice, photos from digital camera are already compressed (in JPEG), and this manual compression with color quantization does not really work. But this technique is used for the standard image format GIF, which uses a color palette of only 256 colors. It is clear that such a compression technique can be useful if the same color palette can be used for several images [Blo08], a for a short movie (showing only one scene) or small animated images, and this is exactly the main purpose of the GIF format.

Note that the SOM, NG, and DSOM algorithm can also be applied to color quantization, and they give the same results, but we only included an illustration for K-Means to keep this report as concise as possible.

---

3 Models of unsupervised learning inspired from neuroscience

After having recalled what is unsupervised learning and the clustering or vectorial quantization problem in the previous section, we present here three clustering algorithms, inspired by neuro-biology or neuroscience: the Self-Organizing Map (SOM, 3.1), the Neural Gas (NG, 3.2), and the (Dynamic) Neural Fields (DNF, 3.3).

3.1 Self-Organizing Maps (SOM)

The first biologically inspired model will take his inspiration from the visual cortex organization. Visual areas in the brain appear to be spatially organized (thanks to unsupervised training), in such a way that physically close neurones in the cortex visual handle input signal physically close in the retina [Koh82].

[7] The script reproducing this experiment is `plot_color_quantization.py`.

[8] See online for the full quality picture and its two compressed versions. The photo is from Heimaey in Iceland.

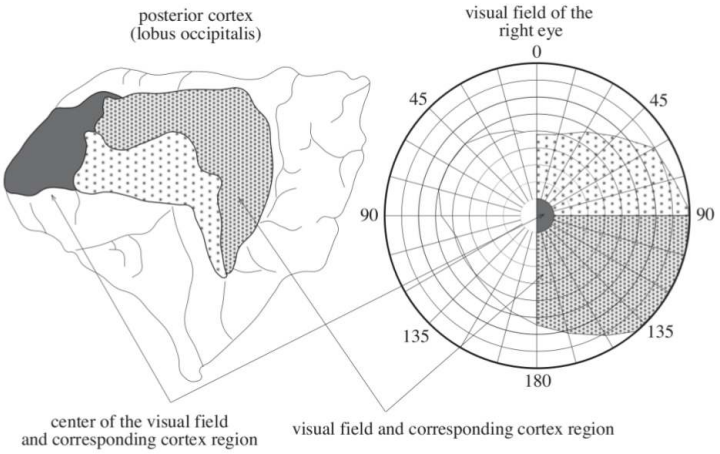


Figure 7: This is referred as "Retinotropic" Organization.

In 1982, from these observations (Fig. 7), T. Kohonen tried to model the spatial organization of the visual cortex [Koh82, Koh98], and by doing so he developed the Self-Organizing Map (SOM) model. A good reference is [Fau94, Part 4.2].

3.1.1 The SOM model

Let start by considering a *map* of n neurons, **fully** inter-connected. We add a *topology* on the map, in $\mathbb{R}^q$, and each neuron i is linked with **all** the input signal (the weight vector $\mathbf{w}_i$ is called the "prototype" of a neuron). Each time a new input data $\mathbf{x}$ is presented, the neuron with the closest prototype wins, and the prototypes of the winner (and his neighbors) are updated, to become closer to the input data. We iterate this step as long as we have training data (or we can cycle back in some cases).

3.1.2 Illustrations for the SOM model

A few figures[9] will help visualizing these assumptions made on the model.

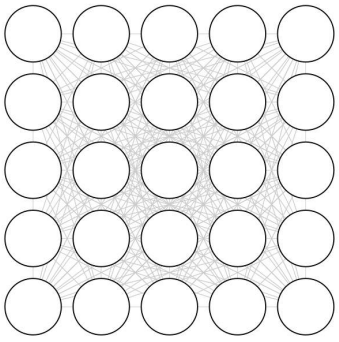
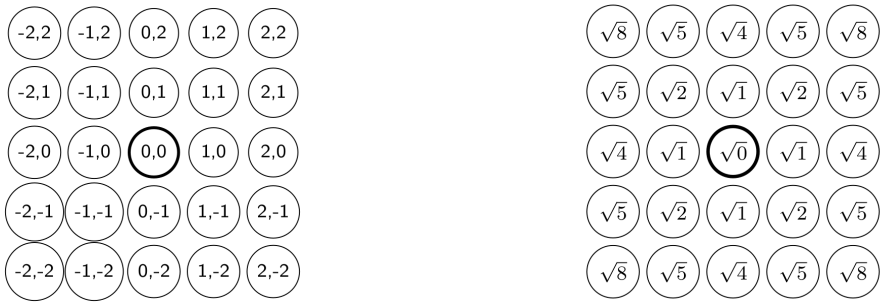


Figure 8: 5×5 fully inter-connected neuronal map.

Each neuron i has a *fixed* position $\mathbf{p}_i$ in $\mathbb{R}^q$ ($q = 2, 3$ usually), but an *evolving* prototype $\mathbf{w}_i$ (lying in the data space E). As soon as we add a *topology* on the map, with natural coordinates in $\mathbb{R}^q$, an inter-neuron Euclidean *distance* $\|\cdot\|$ appears:

[9] They are borrowed from [Rou13].



(a) Natural coordinates. (b) Euclidean distances.

Figure 9: Natural topology on a 5×5 map.

Each neuron is linked with **all** input signals $\mathbf{x}$, as shown below with two inputs $\mathbf{x}_0, \mathbf{x}_1$:

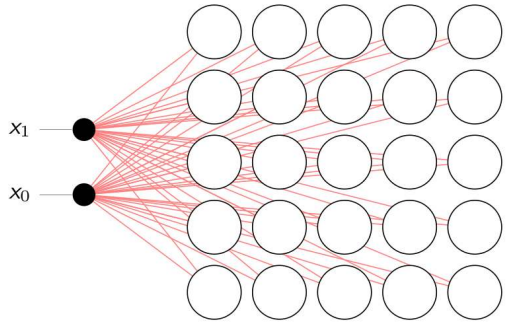


Figure 10: Two inputs $\mathbf{x}_0, \mathbf{x}_1$ for this 5×5 dense neuronal map.

3.1.3 The SOM algorithm

With these notations, the SOM learning algorithm is simply two repeated steps:

1. Choosing the winning neuron: Simply take the index of the neuron minimizing the distances between $\mathbf{x}$ (new input) and the prototypes $\mathbf{w}_i$: $i_{\text{win}} \in \arg\min_{i=1..n} d(\mathbf{x}, \mathbf{w}_i)$.

Remark 3.1. ***Issue with this*** $\arg\min$*: This computation of an* $\arg\min$ *requires a centralized entity, so it is not a* distributed *model. And this is not a very realistic model of cortex organization, as there is no "super-neuron" in the brain in charge of centralized computations. Any realistic model of the cortex has to take into account the highly non-centralized architecture of the brain [Doy00].*

2. Learning step: At each step, a new input $\mathbf{x}$ is given to the neural map, and the winning unit and all its neighbors will update their prototypes to become closer to $\mathbf{x}$, with this vectorial update rule:

$$\mathbf{w}_i(t+1) \leftarrow \mathbf{w}_i(t) + \varepsilon(t) \cdot h(\mathbf{w}_i(t) - \mathbf{x}) \cdot (\mathbf{w}_i(t) - \mathbf{x}) \tag{3}$$

Where $\varepsilon(t) > 0$ refers to a (decreasing[10]) *learning rate* and $h(\cdot)$ is a *neighborhood function*.

The neighborhood function is used in the update rule only with the distance between the sample $\mathbf{x}$ and the winning neuron, so this is a *fully isotropic model*, and this is a satisfactory property if we want to model the cortex, as it is (almost) isotropic [Doy00].

Fig. 11 shows different functions h that can be used as a neighborhood function:

[10] Note that it does not need to go to zero when $t \to \infty$, and in fact t is usually bounded by t_{end}.

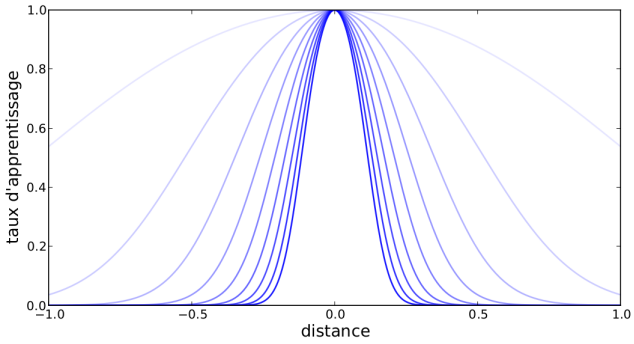


Figure 11: Different learning rate as a function of **distance from the winning neuron**.

3.1.4 Parameters for a SOM

We first need to specify the values taken by the time t: $t = t_{\text{init}} \dots t_{\text{end}}$, it starts at $t_{\text{init}} = 0$, and finishes at $t_{\text{end}} = t_f \in \mathbb{N}^*$. One issue this raises is that the end time t_f has to be decided in advanced, so the basic SOM algorithm cannot offer online learning.

The vectorial update rule has now to be written $\Delta \mathbf{w}_i \overset{\text{def}}{=} \varepsilon(t) \cdot h_\sigma(t, i, i_{\text{win}}) \cdot (\mathbf{x} - \mathbf{w}_i)$.

The learning rate $\varepsilon(t)$, is usually chosen as a *decreasing* function of t. We choose the last and first values $0 \leq \varepsilon_{\text{end}} \ll \varepsilon_{\text{init}}$, and propose to use this simple formula [RB11a, Fau94]:

$$\varepsilon(t) \overset{\text{def}}{=} \varepsilon_{\text{init}} \left(\frac{\varepsilon_{\text{end}}}{\varepsilon_{\text{init}}} \right)^{t/t_f}. \tag{4}$$

A second issue this choice raises is that the map is (almost) fixed after a certain time, so the SOM algorithm will probably fail to follow highly moving dynamic distribution.

And for the neighborhood function h_σ and width $\sigma(t)$, the usual form in the literature is a Gaussian on neuron positions $\mathbf{p}_i$ [Fau94]:

$$h_\sigma(t, i, i_{\text{win}}) \overset{\text{def}}{=} \exp\left(-\frac{\|\mathbf{p}_i - \mathbf{p}_{i_{\text{win}}}\|^2}{2\sigma(t)^2} \right). \tag{5}$$

As for the width $\sigma(t)$, it should also be decreasing, so we choose the last and first values $0 < \sigma_{\text{end}} \ll \sigma_{\text{init}}$, and use the same geometrical formula [Fau94]:

$$\sigma(t) \overset{\text{def}}{=} \sigma_{\text{init}} \left(\frac{\sigma_{\text{end}}}{\sigma_{\text{init}}} \right)^{t/t_f}. \tag{6}$$

3.1.5 Quick theoretical study of the SOM algorithm

By sake of conciseness, this is the only theoretical consideration on an algorithm presented in this report, but a similar analysis could be done for the NG and DSOM algorithms.

Theorem 3.2 (Computational Complexity of SOM algorithm)**.** *Consider a Self-Organizing Map of n neurons, each being q-dimensional, and feed it input data $\mathbf{x}_j$ that are d-dimensional (i.e. all $\mathbf{p}_i \in \mathbb{R}^q$, and all $\mathbf{x}_j \in \mathbb{R}^d$).*

Then each of the t_f learning steps of the SOM algorithm costs about $\mathcal{O}((d + p \times n)n)$ elementary operations.

Therefore, the final theoretical complexity for the SOM algorithm is:

$$\mathcal{O}\left(t_f(dn + pn^2)\right). \tag{7}$$

Proof. When a new training sample $\mathbf{x}$ is presented to the map:

- Computing i_{win} requires to consider all the neurons, taking the $\arg\min$ of distances on neuron positions $\mathbf{p}_j$ (in dimension p), so it takes $\mathcal{O}(pn)$ elementary operations;
- So computing $h_\sigma(t, i, i_{\text{win}})$ requires one computation of i_{win}, and requires to compute a distance of two neuron positions $\mathbf{p}_i$ and $\mathbf{p}_{i_{\text{win}}}$, so $\mathcal{O}(p + pn) = \mathcal{O}(pn)$;
- Computing the vectorial difference $(\mathbf{x} - \mathbf{w}_i)$ takes $\mathcal{O}(d)$ (in $\mathbb{R}^d$);
- So for each neuron of the n neurons, the update rule requires $\mathcal{O}(d + p \times n)$ operations.

□

Note that the dimension q is usually 2 or 3, the size of the neural map n is usually small (8×8 or 32×32 in the experiments below), and the dimension d of the data should never be too big ($d = 2$ or 3 below). The only parameter that will be big is t_f, the number of training samples given to the map, e.g. it is 10000 in the experiments below.

3.2 Neural Gas (NG)

We observed above some weaknesses of this first clustering algorithm (SOM), and it will be our starting point to introduce the Dynamic SOM extension in section 4. Let us quickly present two others models of unsupervised learning, also inspired from neuro-biology, the Neural Gas model and then the Neural Field model.

The Neural Gas model This second model is very similar to a SOM, but it has no underlying topology for the neuron space $\mathbb{R}^q$, we only consider the prototypes $\mathbf{w}_i$.

The same kind of learning algorithm will be used, except that for the Neural Gas, for a new input $\mathbf{x}$, all neurons i are **ordered by increasing distance** of $\mathbf{w}$ to $\mathbf{x}$, and assigned a **rank** $k_i(\mathbf{x})$ (in $1..n$), and this rank is used for updating their prototypes.

The name "gas" comes from this idea: all particles (neurons) influence any other, but the effect of their influence decreases quickly with distance. Note that this idea of using a sort seems counter-intuitive from a biological point of view. As for Self-Organizing Maps and their $\arg\min$, the sorting step for Neural Gas cannot be computed in a decentralized manner, and therefore the NG model is also not a satisfactory model of cortical self-organization [RB11a, Doy00]. And because of the use of this sorting algorithm, there is a unavoidable $\log(n)$ additional factor in the algorithm computational complexity[11].

The update rule for Neural Gas is inspired from the one for SOM, but is modified to use the ranks $k_i(\mathbf{x})$ instead of the distances $\|\mathbf{x} - \mathbf{w}_{i_{\text{win}}}\|$ in the neighborhood $h(\cdot)$:

$$\Delta \mathbf{w}_i \overset{\text{def}}{=} \varepsilon(t) \cdot h_\sigma(t, i, \mathbf{x}) \cdot (\mathbf{x} - \mathbf{w}_i). \tag{8}$$

The same formulas are used for the learning rate $\varepsilon(t)$ and width $\sigma(t)$, both decreasing with time geometrically. But the neighborhood function is now a inverse *exponential on* ***ranks*** (i.e. a Laplace function):

$$h_\sigma(t, i, \mathbf{x}) \overset{\text{def}}{=} \exp\left(-\frac{k_i(\mathbf{x})}{\sigma(t)}\right). \tag{9}$$

[11] In fact, I am not perfectly sure about this point, because we are only sorting the integer indexes list $[1..n]$, so maybe an efficient sorting algorithm (e.g. counting sort) could be used to sort the neurons in a time only $\mathcal{O}(n)$ and not $\mathcal{O}(n \log(n))$. None of the reference articles talk about computational complexity of the algorithms they present, so I have no reference for this question.

The basic Neural Gas algorithm, as presented above, does not allow online learning, and suffer from the same weaknesses as the SOM model. Some extensions to the Neural Gas model has been developed to try to fix this, mainly the Growing NG or Dynamic NG models.

By lack of both time and space, we do not cover the Neural Gas model more in details, for all the details see [Fau94] for the basic model, and [RB11a] for a unified point of view and the NG model presented with the same notations as the SOM/DSOM model.

3.3 Dynamic Neural Fields (DNF)

In this subsection, we present very quickly the Dynamic Neural Field (DNF) model, based on its presentation in [RD11], without giving more details. **Neural Fields** are another family of models, inspired from continuous LeapField models (from EEG), rather than neural networks, for instance see [Fau94] for an old-fashion presentation of the classical model. N. Rougier and G. Detorakis developed in 2011 an extension of the NF model to make it work on dynamic distribution, and to also model self-organization.
They considered a membrane potential V, which follows this *functional PDE*:

$$\tau \frac{\partial V(\mathbf{x},t)}{\partial t} = -V(\mathbf{x},t) + h + I(\mathbf{x},t) + \int_M W(\|\mathbf{x}-\mathbf{y}\|) \cdot f(V(\mathbf{y},t)) \, \mathrm{d}\mathbf{y}. \tag{10}$$

Where $V(\mathbf{x},t)$ is the membrane potential at position $\mathbf{x} \in E$ and at time t (continuous in paragraph), $W(\|\mathbf{x}-\mathbf{y}\|)$ is the lateral connection weight between $\mathbf{x}$ and $\mathbf{y}$; f is the mean firing rate, h is the resting potential; and $I(\mathbf{x},t)$ is the input at position $\mathbf{x}$. Then the PDE is solved with a numerical discretization, and a simple forward Euler scheme.

In order for this DNF solution to also models self-organization, they considered a new learning rule, slightly different from one the used by the SOM and NG algorithms:

- If a neuron is "*close enough*" to the data, there is no need for others to learn anything: the winner can represent the data alone;
- If there is no neuron close enough to the data, any neuron learns the data according to its own distance to the data.

We conclude here section 3, after having presented the SOM, NG and DNF models.

---

4 Dynamic Self-Organizing Maps (DSOM)

For this last section, we go back to the SOM model, and after summing up a few of its strengths and weaknesses as presented above, we introduce Dynamic Self-Organizing Maps, as a simple extension of SOM. We conclude by presenting illustrated results of some experiments, comparing SOM, NG and DSOM, and also demonstrating the effects of the two hyper-parameters of a DSOM (elasticity and learning rate).

4.1 What need for a dynamic model?

As exposed above, the SOM model has some weaknesses,

- The map topology can not correspond to the data topology, this can ruins the learning possibility (e.g. if the initialization messed up).

- The map can fail to deploy correctly in the first learning steps, and we get big aggregates of prototypes (we fall into a local minimum of distortion).
- The map is fixed after training, as learning rate goes to $\varepsilon_{\text{end}} \ll 1$ (no long-term learning, i.e. only stationary distributions). This has the advantage of modeling parts of the learning process in early years (in child), but implies that a SOM cannot model long-term or life-long learning (in adult).
- And similarly, we have to know the ending learning time t_f in advance, i.e. the number of training examples given to the map, so the SOM model cannot be used for online learning.

4.2 Constant learning rate on a SOM

Based on these observations and the few weaknesses of the SOM model, N. Rougier and Y. Boniface has proposed in [RB11a] an extension called the Dynamic Self-Organizing Maps (DSOM). To obtain the DSOM model, we simply need to change the update rule $\Delta \mathbf{w}_i$, and neighborhood function. At each new input data $\mathbf{x}$, update the winning prototype (and its neighbors):

$$\Delta \mathbf{w}_i \overset{\text{def}}{=} \varepsilon_0 \cdot \|\mathbf{x} - \mathbf{w}_i\|_E \cdot h_\eta(i, i_{\text{win}}, \mathbf{x}) \cdot (\mathbf{x} - \mathbf{w}_i). \tag{11}$$

Where $\varepsilon_0 > 0$ is now a *constant* learning rate, $\eta > 0$ is a *elasticity / plasticity parameter*, and h_η is a time-invariant neighborhood[12] function:

$$h_\eta(i, i_{\text{win}}, \mathbf{x}) \overset{\text{def}}{=} \exp\left(-\frac{1}{\eta^2} \frac{\|\mathbf{p}_i - \mathbf{p}_{i_{\text{win}}}\|^2}{\|\mathbf{x} - \mathbf{w}_{i_{\text{win}}}\|_E^2}\right). \tag{12}$$

Remark 4.1. ***Interpretation****: It is like having time-invariant but* local dependent *learning rate ε and width σ. This implies that both parameters will (virtually) behave like if they were $\tilde{\sigma} \overset{\text{def}}{=} \eta \cdot \|\mathbf{x} - \mathbf{w}_s\|_E$. $\tilde{\varepsilon} \overset{\text{def}}{=} \varepsilon_0 \cdot \|\mathbf{x} - \mathbf{w}_s\|_E$. The closer the winning prototype $\mathbf{w}_{i_{win}}$ is to $\mathbf{x}$, the smaller both $\tilde{\sigma}, \tilde{\varepsilon}$ will be, and the smaller the update on the prototypes will be; and conversely. This is a very logical learning rule: no need to update $\mathbf{w}_{i_{win}}$ a lot if it already represents well the input $\mathbf{x}$, but the prototype very far away should be more affected.*

There is several consequences of having a constant learning rate:

- **Online learning**, as there is no need to specify an end time t_f, the map can accept data as long as needed.
- And **long-term learning**, $\varepsilon(t)$ does not $\to 0$ with $t \to \infty$, so the map can still evolve as long as necessary in the future.
- **Less parameters**: instead of 5 parameters (t_f, σ_{init}, σ_{end}, $\varepsilon_{\text{init}}$, ε_{end}) for the SOM algorithm, the DSOM algorithm only needs 2 parameters, a constant learning rate ε_0 and an elasticity η.
- But convergence seems harder, and stability is not achievable, so DSOM has less theoretical guarantee.

[12] As a convention, we ask $h_\eta(i, i_{\text{win}}, \mathbf{x}) \overset{\text{def}}{=} 0$ if $\mathbf{x} = \mathbf{w}_{i_{\text{win}}}$ to avoid the pole, and to get a continuous neighborhood function h_η.

4.3 Comparisons between NG, SOM, DSOM

In this subsection, we present a series of numerical experiments done with programs written in Python, to implement and compare the NG, SOM and DSOM models. The programs used were borrowed from [RB11b].

The experiment setup was quite simple: Three networks (NG, SOM, DSOM) of $n = 8 \times 8$ nodes (in $\mathbb{R}^2$) are trained for $t_f = 20000$ iterations, on various distributions f on a $2D$ square $[0,1] \times [0,1]$. The initialization for prototypes $\mathbf{w}_i$ is purely random (uniform on the square). Decreasing distortion $\mathbf{J}$ is showed as function of training time above the final codebook map. It is always decreasing.

In all the following figures, the small blue points are the training samples $\mathbf{x}_j$, and the big **white points** are the vectors of the codebook $\mathbf{w}_i$. A short observation is given along each figure.

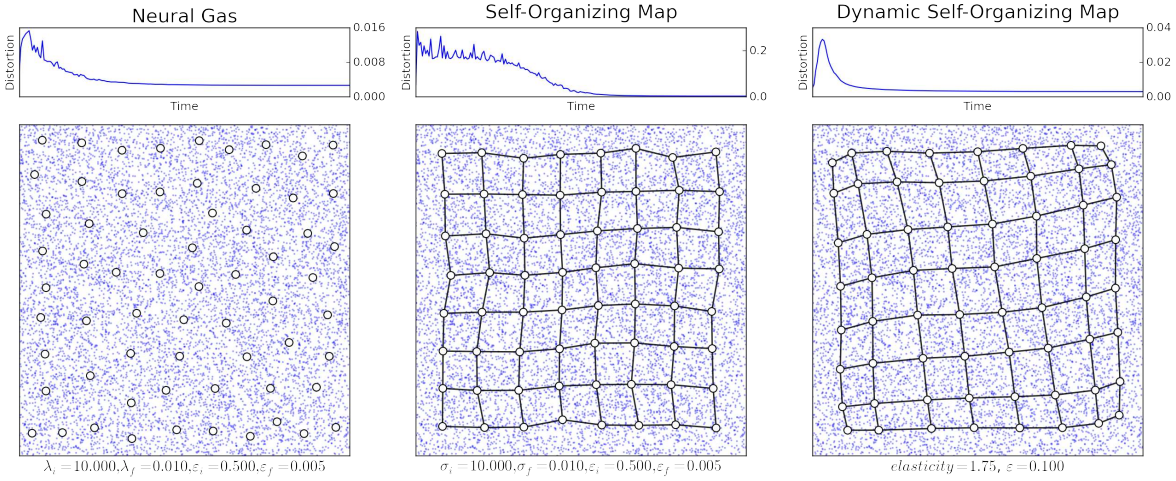


Figure 12: A simple uniform distribution.

– We observe in Fig. 12 that DSOM gives a smoother map than SOM, and this is true in general (for well chosen parameters).

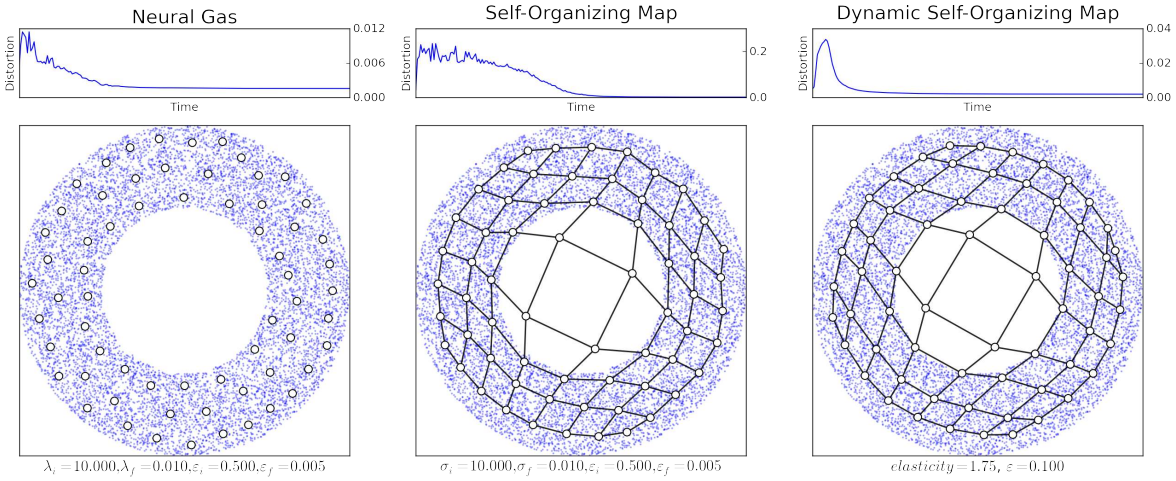


Figure 13: A simple ring distribution.

– We observe in Fig. 13 that the distortion usually decreases more quickly and smoothly

with a DSOM than a NG/SOM. This is a very nice property, as it indicates a quicker convergence.

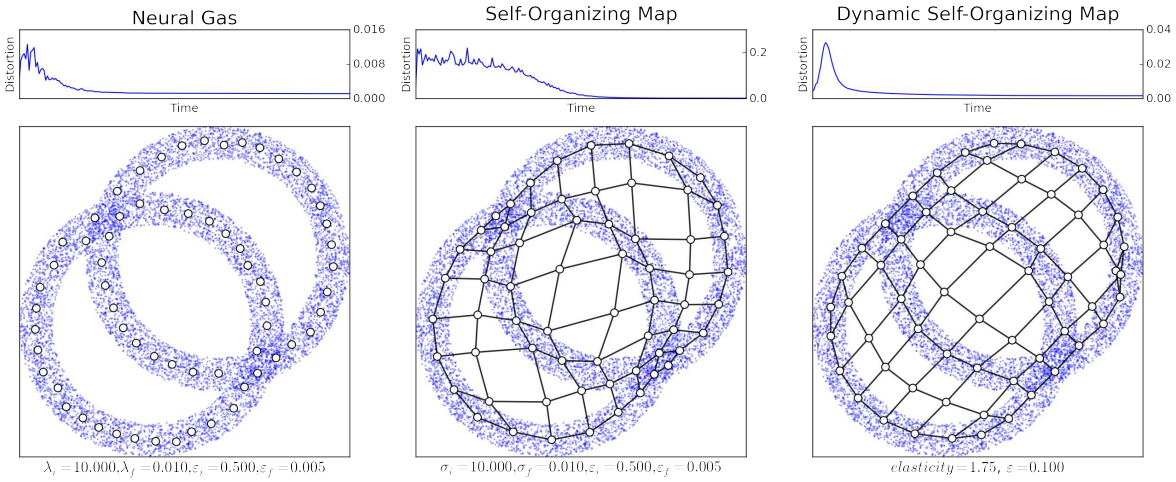


Figure 14: Double ring distribution.

– We observe in Fig. 14 that on this double ring distribution, the Neural Gas achieves a way lower final distortion, and this is logical as both the SOM and the DSOM will be forced to use nodes in the middle of the two rings, where the target density is null. This is one example where we would obtain a better final map if the SOM/DSOM model were fitting the magnification law.

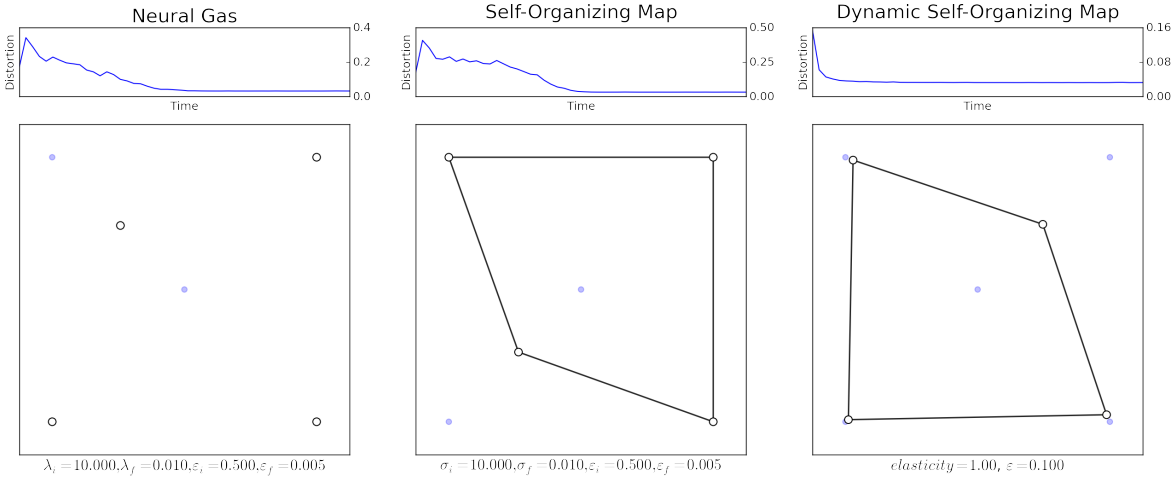


Figure 15: Issue for wrongly designed topology: 4 nodes for 5 data points.

– We observe in Fig. 15 that if the topology is not correctly designed, e.g. as here if there is only 4 nodes but 5 data points, the neural map models perform worse than the Neural Gas.

– In Fig. 16 is considered a first non-stationary distribution on the square, we considered a uniform distribution on a quarter, moving after each 5000 iterations: first $[0, 0.5] \times [0, 0.5]$ (3), then $[0.5, 1] \times [0.5, 1]$ (2), then $[0, 0.5] \times [0.5, 1]$ (1), and finally $[0.5, 1] \times [0, 0.5]$ (4). The Neural Gas only has a short-term learning ability, and we see most of its neurons are stuck in the first quarter (3). the SOM does a better job, but the parameters $\varepsilon_{\text{end}}, \sigma_{\text{end}}$ are probably too small or t_f is too small, and the SOM stops learning before the last move of the dynamic

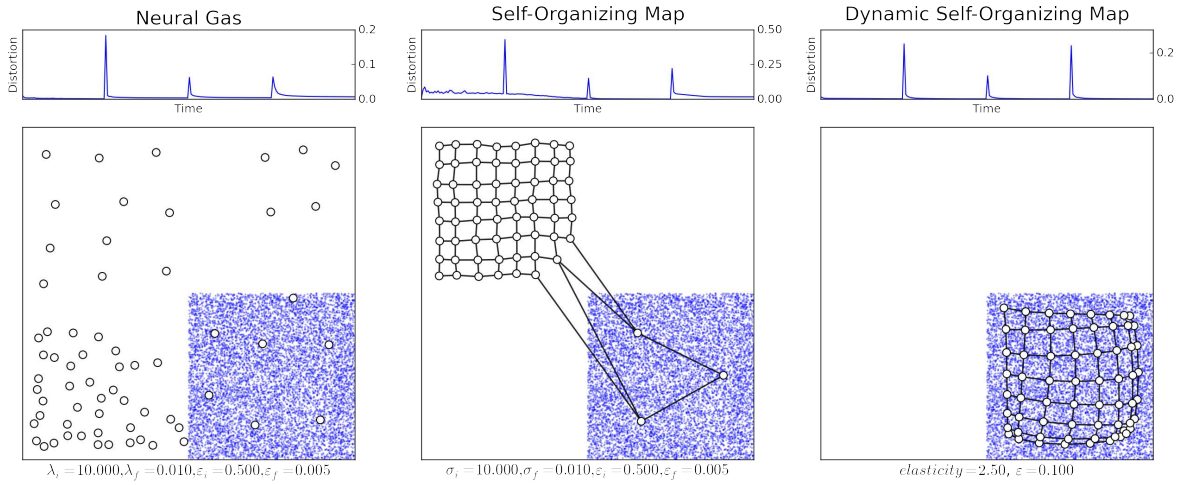


Figure 16: Non-stationary distribution, moving from quarters $3 \to 2 \to 1 \to 4$, at regular epochs.

distribution: most of its neurons are in the quarter (1). And the DSOM is extremely versatile here, and it allows **long-term learning** (all its neurons followed each of the 4 quarters). This observation allows to affirm that the DSOM algorithm can model the cortical plasticity as a tight coupling between model and environment.

4.4 Examples of non-stationary distributions

We finish with a last example of a DSOM applied to several distributions. First, it is applied on a stationary distributions (a cube or a sphere in $3D$), and then on two non-stationary distribution: a $2D$ manifold continuously changed from a sphere to a cube, or conversely from a cube to a sphere.

The experimental setup is similar to the first experiment. A DSOM with $n = 32 \times 32$ nodes (in $\mathbb{R}^3$) has been trained for $t_f = 10000$ iterations. On a set of 10000 points uniformly distributed over the surface of a sphere or a cube of radius 0.5 centered at $(0.5, 0.5, 0.5)$ in $\mathbb{R}^3$. Initialization has been done by placing initial code vectors at the center of the sphere. The elasticity parameter η has been set to 1. We observe self-organization on a sphere or cubic surface, or self-reorganization from sphere to cubic surface (or inverse).

The results of these 4 experiments are short animations (about 50 seconds), all availables online on `http://www.labri.fr/perso/nrougier/coding/article/article.html`.

4.5 Questions still not answered

Magnification law for a DSOM ? One observation we made about the K-Means algorithm was that it fit the magnification law, and the same is true for both the Neural Gas and the Neural Field models. But we can observe on Fig. 18 that both the SOM and DSOM algorithm do not fit the "magnification law": the resulting map is almost exactly the same[13] for the three different densities.

[13] Up to a random rotation – which is a very satisfactory observation. Indeed, we notice that if the distribution f is isotropic, the SOM and DSOM models are also isotropic, i.e. no particular direction is favored, so in case of finite codebook an alignment direction is randomly picked by the map at the beginning of the training process (and this is always true for these kinds of maps).

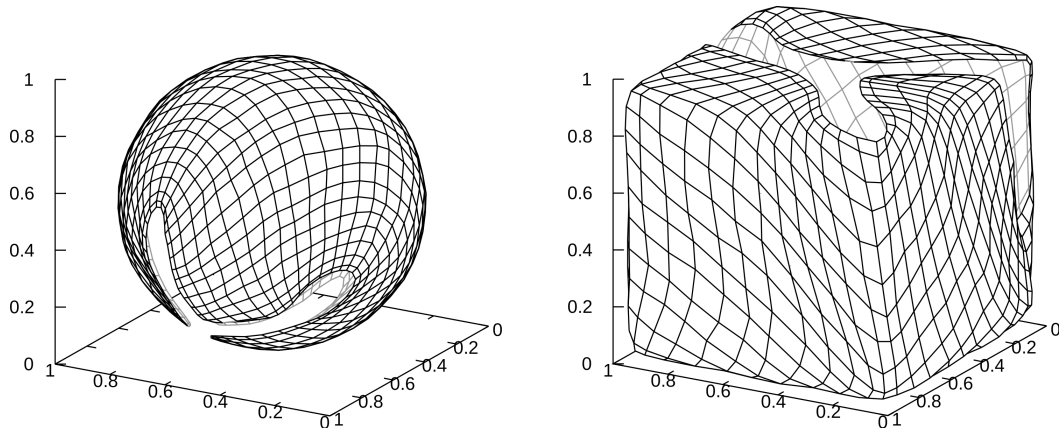


Figure 17: Non-stationary distribution: a DSOM going from a sphere to a cube distribution.

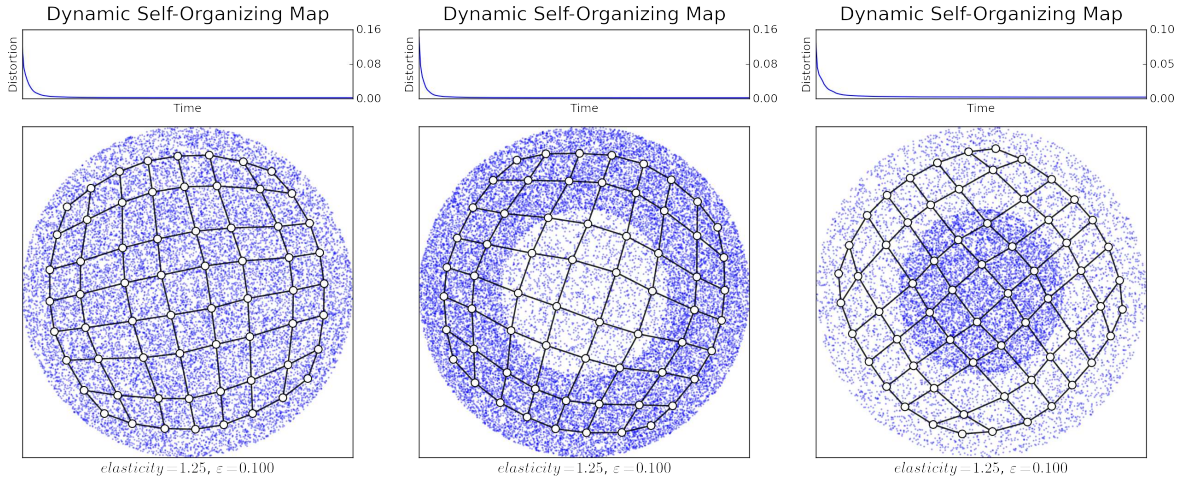


Figure 18: DSOM is invariant regarding local density of the target distribution f.

But is it good news or bad news? Well this depends[14] on the context where the SOM/DSOM is used, for some distribution we would like to have a denser prototypes density where the empirical density is more important, but for other cases we would like to keep a pseudo-uniform prototypes density.

Elasticity The influence of the elasticity parameter can be understood from the Fig. 19 below. A more detailed analysis of this parameter η is given in [RB11a, Sec. 3.2].

A natural question is: can we find a way to automatically tune the elasticity or width parameter? (σ_{init} and σ_{end} for SOM, or η for DSOM).

It seems hard to do, because in this unsupervised setting, the only measure of quality we can get from the resulting map is the distortion, and as the above Fig. 19 shows, three different values of η can give completely different distribution of the final prototypes of the DSOM map, but exactly the same final distortions (and very similar evolution of the distortions).

If we quit the fully unsupervised world, and allow to "visually judge" the quality of the final maps, then we can select the best η among the list of explored parameters (here, it

[14] I have not worked more on this question.

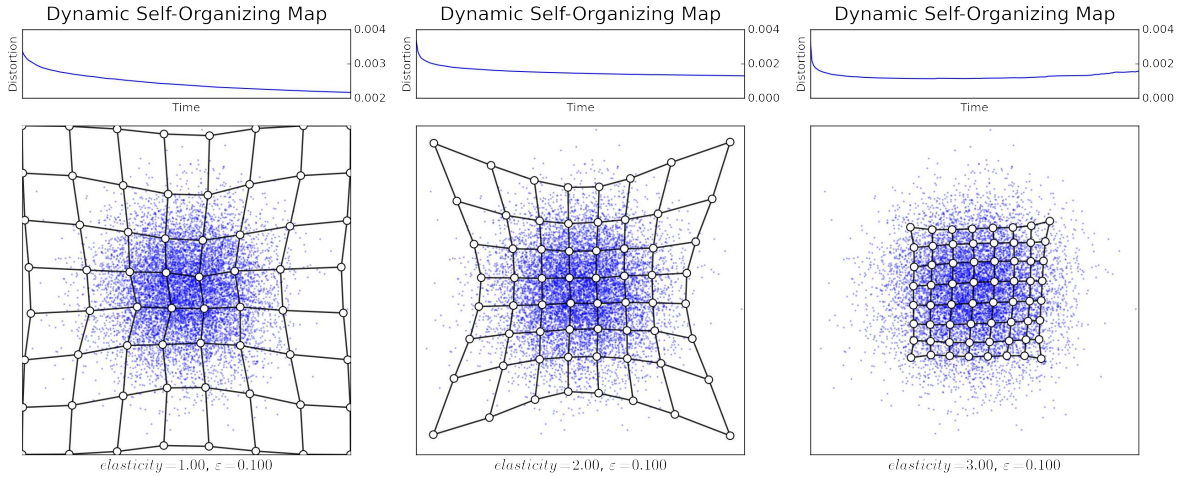


Figure 19: Influence of the elasticity parameter η (3 DSOM: $\eta = 1, 2, 3$).

seems that $\eta = 3$ gives the best final quantization). So one has to be cautious that any kind of grid search (either for elasticity or width), based only on the distortion criteria, *cannot* be enough to select the best value.

Harder questions These harder questions are included here but have not been answered, mainly because of limited time:

– What topology to adopt for higher dimension data (what if $d \geq 2, 3$) ? Example of image processing with NG/SOM/DSOM In [RB11a], an example of use of a three maps (NG/SOM/DSOM) on a higher dimensional dataset is presented. It consists in a vectorial quantization tasks, not on the color-space of an image ($d = 2$) on a similarity graph from small patches of an image ($d = 8 \times 8 = 64$).

– If there is a need for a *topological rupture* (for instance if the distribution f can clearly be split in some sub separate distributions): how to let a DSOM decides to split in 2 (or more) sub-maps? We could imagine a strategy similar to a graph cut, when too many neurons are far from another group of neurons, we cut all the links between the two blobs, and restart from there with two separate DSOM.

– And the worse unanswered question is about the theoretical warranties of all these unsupervised algorithms. One have to keep in mind that both convergence and stability has *not been proved* for any of the SOM, NG and DSOM algorithm, as well as K-Means (for just one random initialization, there is always a chance of finding a local minimum). The Self-Organized DNF model as presented in [RD11] has a better warranty of convergence. But stability even seems unachievable if we want to keep long-term learning (online learning).

5 Conclusion

In this report, we started by recalling what are the different types of learning, found in machine learning and also found in the brain, as some neuro-biology experiments showed. Supervised learning and reinforcement already are applied on daily technology, and so is unsupervised learning, but even if the latter is still the harder, it is the main future direction of research in machine learning these days.

One example of unsupervised task is data clustering, and we explained why clustering algorithms can be useful, by first applying the K-Means algorithm to color quantization for photos compression, and then by studying several biology-inspired models of self-organization and learning (Neural Gas, Neural Fields, SOM and DSOM); which all model correctly some aspect of these two properties of the brain.

Based on numerical experiments comparing NG, SOM and DSOM, we showed the strength and weaknesses of the 3 models, and detailed why a dynamic model can be useful. DSOM takes the advantage by not requiring a centralized computation, having less parameters, and allowing for online and long-term learning, on both static and dynamic distributions. So DSOM appears as an effective model of self-organization in the brain as well as cortical plasticity, modeling the life-long learning process observed in adults. But, despite their weaknesses (the main one being the need for a centralized computation unit), both the NG and SOM models are satisfying models of self-organization and the early-year learning process observed in child.

Experimentally, we also applied K-Means and a SOM for color quantization and image compression, on several examples; then we compared NG, SOM and DSOM on several stationary and non-stationary distributions in 2D [RB11a]; and we also compared SOM and DSOM on a higher dimension distribution (from image processing) [RB11a]. And all experiments confirmed the different intuitions we had about the models.

Some theoretical question has been answered, along with two statements on the computational complexity of the SOM and DSOM algorithms, but some theoretical and practical questions are still to be answered, including: how to automatically choosing elasticity η? does a (D)SOM always converges? is a self-organized map stable? We conclude this report with these open questions, hoping to be able to come back on them and finish this work in the future.

A Appendix

A.1 Acknowledgments

I would like to thank Nicolas P. Rougier as he replied quickly to my initial queries and provided useful initial directions of research; and Jean-Pierre Nadal (the professor in charge of the course) for an interesting course on neuroscience and modeling aspects in neuroscience.

A.2 Personal feeling about the project

I enjoyed working on this small project, and as usual for this kind of maths, I liked the different aspects we touched with this project: algorithms, algorithm complexity proofs, implementations, numerical simulations, models inspired from neuroscience or biology, etc. With more time, I would have liked to try to apply the algorithms presented here on a real-world problem or on a more complete neuroscience experiment; and I would have liked to try longer to answer the harder questions.

This work is dedicated to the memory of Nicolas Pajor. ☹

A.3 References

[Bis06] Christopher M Bishop (2006). *Pattern Recognition and Machine Learning*. Springer.

[Blo08] Dan S. Bloomberg (2008). *Color quantization using octrees*. URL `http://www.leptonica.com/color-quantization.html`, online tutorial, published on www.leptonica.com.

[CFP87] Marie Cottrell, Jean-Claude Fort, and Gilles Pagès (1987). *Étude d'un algorithme d'auto-organisation*. *Annales de l'Institut Henri Poincaré*, 23(1):1–20.

[CFP98] Marie Cottrell, Jean-Claude Fort, and Gilles Pagès (1998). *Theoretical Aspects of the SOM Algorithm*. *Neurocomputing*, 21(1):119–138. URL `http://arxiv.org/abs/0704.1696v1`.

[DK03] Jeremiah D. Deng and Nikola K. Kasabov (2003). *On-line Pattern Analysis by Evolving Self-Organizing Maps*. *Neurocomputing*, 51:87–103.

[Doy00] Kenji Doya (2000). *Complementary roles of basal ganglia and cerebellum in learning and motor control*. *Current opinion in NeuroBiology*, 10(6):732–739.

[Fau94] Laurene Fausett (1994). *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. ISBN 0-13-334186-0.

[Koh82] Teuvo Kohonen (1982). *Self-Organized Formation of Topologically Correct Feature Maps*. *Biological Cybernetics*, 43(1):59–69.

[Koh98] Teuvo Kohonen (1998). *The Self-Organizing Map*. *Neurocomputing*, 21(1):1–6.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). *Scikit-learn: Machine Learning in Python*. *Journal of Machine Learning Research*, 12:2825–2830. URL `http://scikit-learn.org/dev/about.html`.

[RB11a] Nicolas P. Rougier and Yann Boniface (2011). *Dynamic Self-Organizing Map*. *Neurocomputing*, 74(11):1840–1847. URL `https://hal.inria.fr/inria-00495827`.

[RB11b] Nicolas P. Rougier and Yann Boniface (2011). *Dynamic Self-Organizing Map*. URL `http://www.loria.fr/~rougier/DSOM/dsom.tgz`, python code sources.

[RD11] Nicolas P. Rougier and Georgios Detorakis (June 2011). *Self-Organizing Dynamic Neural Fields*. In Springer, editor, *International Conference on Cognitive Neurodynamics*, volume III of *Advances in Cognitive Neurodynamics*. Niseko village, Hokkaido, Japan. URL `https://hal.inria.fr/inria-00587508`.

[Rou13] Nicolas P. Rougier (2013). *Dynamic Self-Organizing Map*. URL `http://www.labri.fr/perso/nrougier/coding/article/article.html`, slides for a course on self-organization.

[SB98] Richard S. Sutton and Andrew G. Barto (1998). *Reinforcement Learning: An Introduction*, volume 1. MIT Press, Cambridge, MA. URL `http://webdocs.cs.ualberta.ca/~sutton/book/the-book.html`.

(*Note:* a more detailed bibliography is available on-line, in HTML, PDF and BibTEX.)

License?