Self-Organizing Map (SOM) and Dynamic SOM:
From unsupervised clustering to models of cortical plasticity

Project Presentation – Neuroscience course

Lilian Besson

École Normale Supérieure de Cachan (Master MVA)

March 31st, 2016 | Time : 20 + 10 minutes

Everything (slides, report, programs) is open-source at
http://lbo.k.vu/neuro2016

If needed: lilian.besson@ens-cachan.fr

Grade: I got 17.5/20 for my project.

Topic of the project

Unsupervised learning ?

In machine learning, and in the brain [Doya, 2000], there is:

- Supervised learning (cerebellum);
- Reinforcement learning (basal ganglia and thalamus);
- **Unsupervised learning** (cortex).

Topic of the project

Unsupervised learning ?

In machine learning, and in the brain [Doya, 2000], there is:

- Supervised learning (cerebellum);
- Reinforcement learning (basal ganglia and thalamus);
- **Unsupervised learning** (cortex).

Different unsupervised learning models

- K-Means: a classical one.

Topic of the project

Different unsupervised learning models

- K-Means;
- **Self-Organizing Maps & Dynamic SOM**;
- Neural Gas;
- Neural Field & Dynamic NF.

Topic of the project

Different unsupervised learning models

- K-Means;
- **Self-Organizing Maps & Dynamic SOM**;
- Neural Gas;
- Neural Field & Dynamic NF.

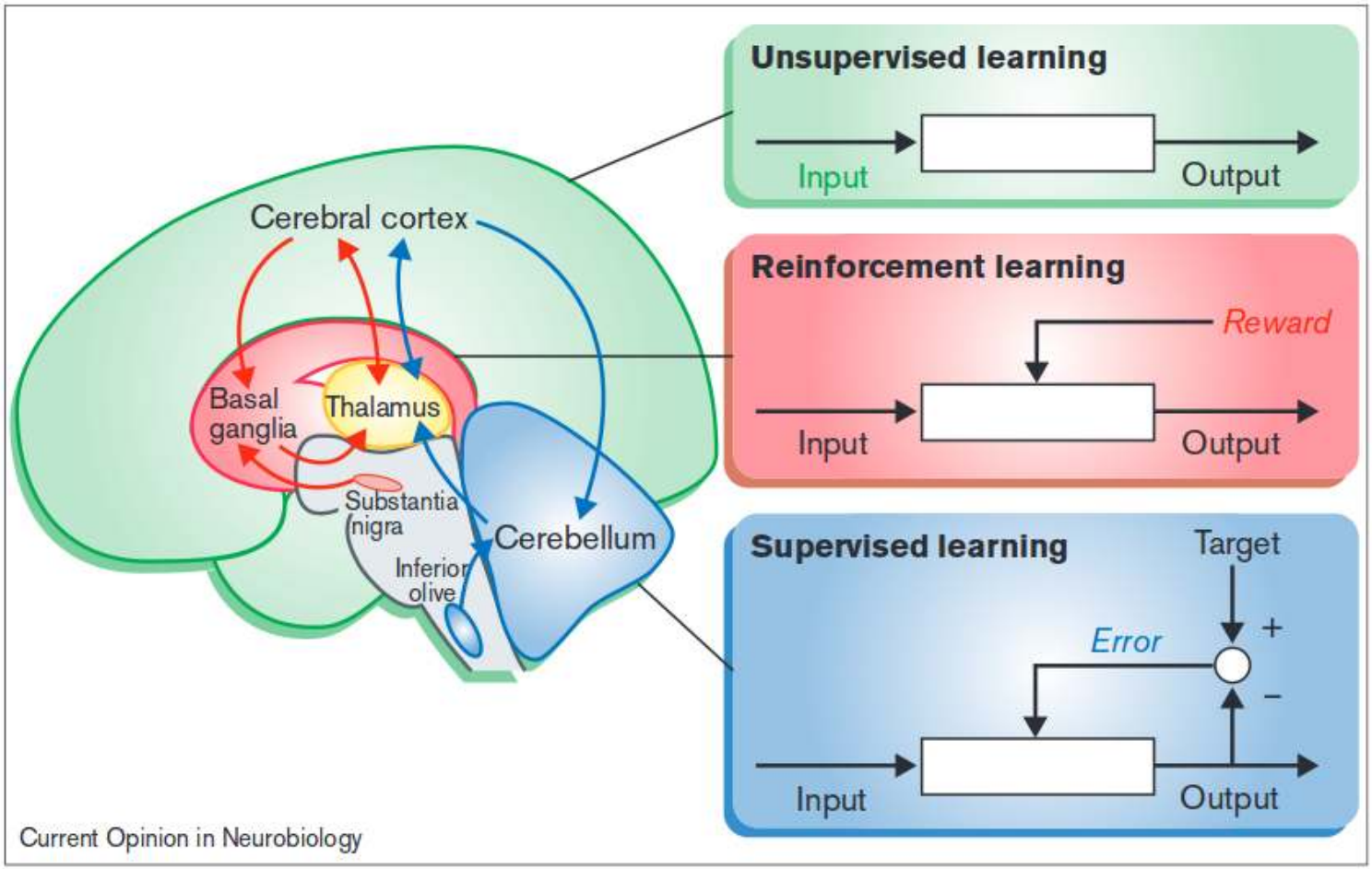
Applications and experiments

1. Data/image compression (e.g. color quantization, GIF);
2. Modeling self-organization and online learning (plasticity) in the cortex;
   - etc.

Outline

1. Introduction & Motivations
2. Unsupervised Learning, starting with K-Means
3. Unsupervised models inspired from neuroscience
4. Dynamic Self-Organizing Maps (DSOM)
5. Conclusion & Appendix

Learning in the brain



The 3 main types of learning are present in the brain [Doya, 2000, Figure 1].

In Machine Learning : supervised learning I

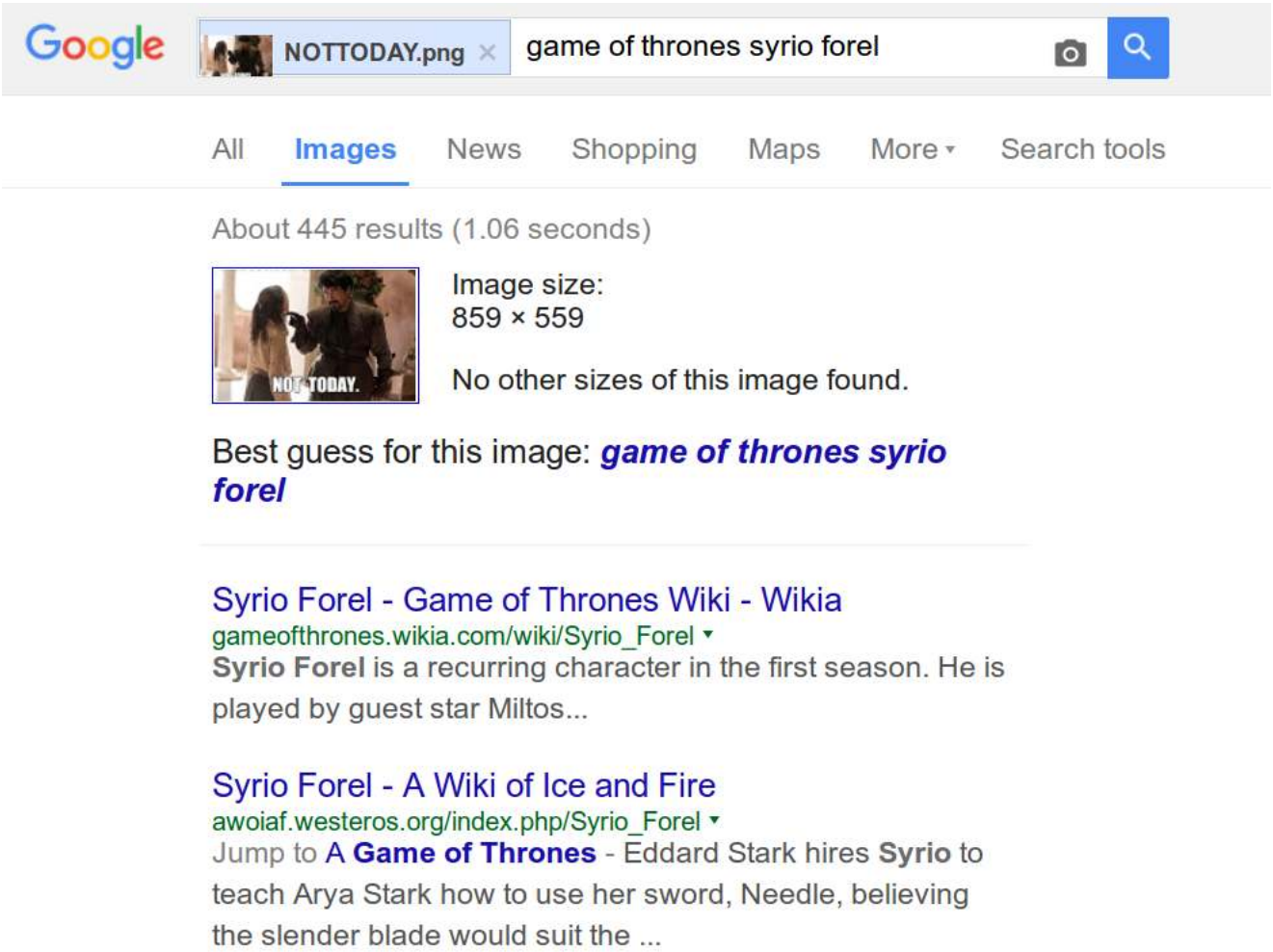
Each type of learning have been studied from the $50'$s.

Supervised/Deep learning [Bishop, 2006]

$\equiv$ Learning from labeled data.

Success story:
Google Images (`images.google.com`) showed that real-world image retrieval works (in 2012).

In Machine Learning : supervised learning II



Deep Learning success: Google Images.

In Machine Learning : reinforcement learning I

Reinforcement learning [Sutton and Barto, 1998]

$\equiv$ Learning with feedback (reward/penalty).

Success story:

Google DeepMind's Alpha Go showed that reinforcement learning (and deep learning) can give powerful AIs (in 2016).

In Machine Learning : reinforcement learning II

Reinforcement Learning success: Google DeepMind's Alpha Go.

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

Why is unsupervised learning harder?

No idea what the data is: no labels, no time organization, no feedback/reward/penalty:

Just **raw** data.

Why is unsupervised learning harder?

No idea what the data is: no labels, no time organization, no feedback/reward/penalty:

Just **raw** data.

Predictive learning is **the future**

A very recent quote from Richard Sutton[a] and Yann LeCun[b]:

> *"AlphaGo is missing one key thing: the ability to learn how the world works."* ***Predictive (unsupervised) learning*** *is one of the things some of us see as the next obstacle to better AI.*
>
> (Yann LeCun quoting Richard Sutton in February 2016)

[a] One of the father of reinforcement learning, cf. [Sutton and Barto, 1998].
[b] One of the father of deep learning.

Vectorial quantization: a simple unsupervised task

Let $X = \{x_1, \ldots, x_p\}$ be samples in a space E:

Goals

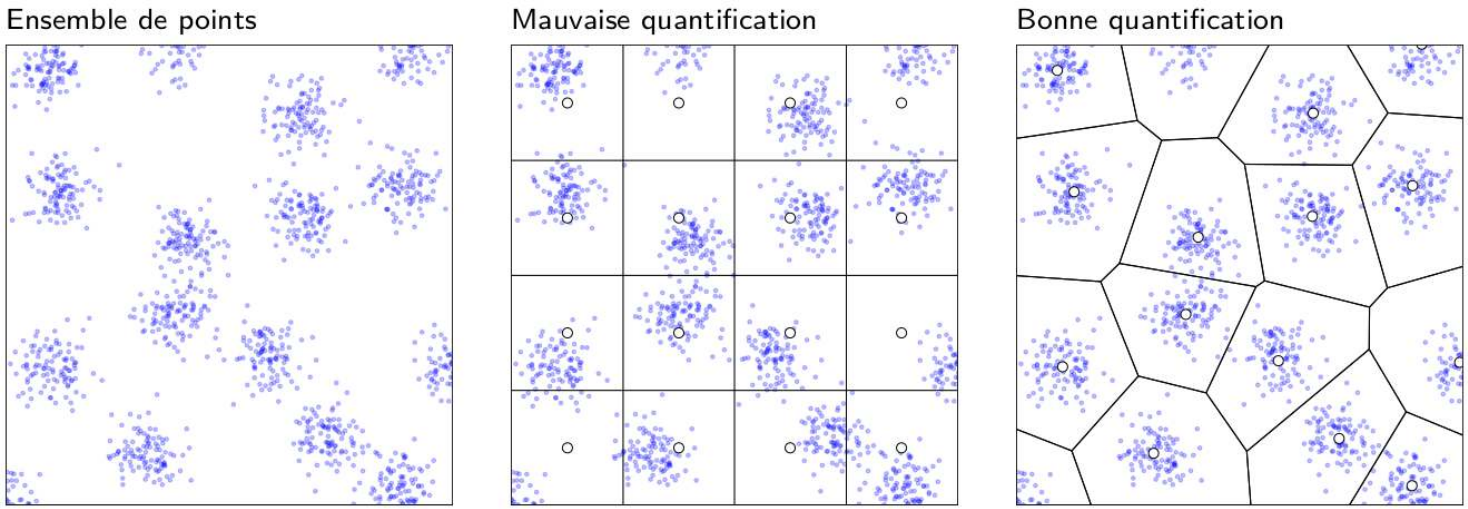
- How to cluster similar data together? Similar in what sense?
- How many groups there is? K clusters C_j: find K.
- What are the best representatives of each group? "Centroids" μ_j.
- Can we identify close groups (and merge them) ?

Vectorial quantization: a simple unsupervised task

Let $X = \{x_1, \ldots, x_p\}$ be samples in a space E:

Goals

- How to cluster similar data together? Similar in what sense?
- 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 2D points, examples of a bad quantization and a good quantization

Notations and objectives of VQ

Definition of a **vectorial quantization** algorithm

Let E be the data space ($X \subset E$), a compact manifold in $\mathbb{R}^d$.

A vectorial quantization of E is defined by a function Φ, and a set $Q \subset E$, so that $\forall \mathbf{x} \in E, \Phi(\mathbf{x}) \in Q$.

Notations and objectives of VQ

Definition of a **vectorial quantization** algorithm

Let E be the data space ($X \subset E$), a compact manifold in $\mathbb{R}^d$.

A vectorial quantization of E is defined by a function Φ, and a set $Q \subset E$, so that $\forall \mathbf{x} \in E, \Phi(\mathbf{x}) \in Q$.

Q is usually discrete/finite, called the *codebook*: $Q = \{w_1, \ldots, w_n\}$.

Notations and objectives of VQ

Definition of a **vectorial quantization** algorithm

Let E be the data space ($X \subset E$), a compact manifold in $\mathbb{R}^d$.

A vectorial quantization of E is defined by a function Φ, and a set $Q \subset E$, so that $\forall \mathbf{x} \in E, \Phi(\mathbf{x}) \in Q$.

Q is usually discrete/finite, called the *codebook*: $Q = \{w_1, \ldots, w_n\}$.

Two examples in $1D$

For data in $E = \mathbb{R}$, if we want to quantize them in Q:

$Q = \{\pm 1\}$: take $\Phi(\mathbf{x}) = \text{sign}(\mathbf{x})$, $\Longrightarrow$ 2 prototypes.

$Q = \mathbb{Z}$: take $\Phi(\mathbf{x}) = \lfloor \mathbf{x} \rfloor$, $\Longrightarrow$ ∞ prototypes.

Notations and objectives of VQ

Definition of a **vectorial quantization** algorithm

Let E be the data space ($X \subset E$), a compact manifold in $\mathbb{R}^d$.

> *A vectorial quantization of E is defined by a function Φ, and a set $Q \subset E$, so that $\forall \mathbf{x} \in E, \Phi(\mathbf{x}) \in Q$.*

Q is usually discrete/finite, called the *codebook*: $Q = \{w_1, \ldots, w_n\}$.

Can we **generalize** to any data?

Find automatically the target/compressed set Q, and the clustering function Φ, for any dataset X in a set E ?

Notations and objectives of VQ

Notations and objectives

- Cluster: $C_i \overset{\text{def}}{=} \{\mathbf{x} \in E : \Phi(\mathbf{x}) = \mathbf{w}_i\}$;
- Target probability density f on E;
- (Continuous) Distortion of the VQ: $\mathbf{J}(\Phi) \overset{\text{def}}{=} \sum_{i=1}^{n} \mathbb{E}_{f,C_i}\left[\|\mathbf{x} - \mathbf{w}_i\|^2\right]$;
- But f is unknown: only p unbiased observations $\mathbf{x}_j$ are available:
  Empirical distortion $\hat{\mathbf{J}}(\Phi) \overset{\text{def}}{=} \frac{1}{p} \sum_{i=1}^{n} \sum_{\mathbf{x}_j \in C_i} \|\mathbf{x}_j - \mathbf{w}_i\|^2$;

$\Longrightarrow$ Goal: minimize the empirical distortion $\hat{\mathbf{J}}$!

A "classical" problem

Several algorithms:

- (1) K-Means;
- Elastic Net (L1-L2 penalized least-squares);
- (2) **(Dynamic) Self-Organizing Map**; [Rougier and Boniface, 2011a]
- (3) (Growing/Dynamic) Neural Gas;
- (4) (Dynamic) Neural Field. [Rougier and Detorakis, 2011]

A "classical" problem

Several algorithms:

- (1) K-Means;
- Elastic Net (L1-L2 penalized least-squares);
- (2) **(Dynamic) Self-Organizing Map**; [Rougier and Boniface, 2011a]
- (3) (Growing/Dynamic) Neural Gas;
- (4) (Dynamic) Neural Field. [Rougier and Detorakis, 2011]

Several applications:

- Compression of data (images etc);
- Automatic classification/categorization[a] etc.

[a] Success story: Netflix "automatically" discovered the main genres of movies in 2013 from its database of movies ratings.

K-Means: a first unsupervised algorithm

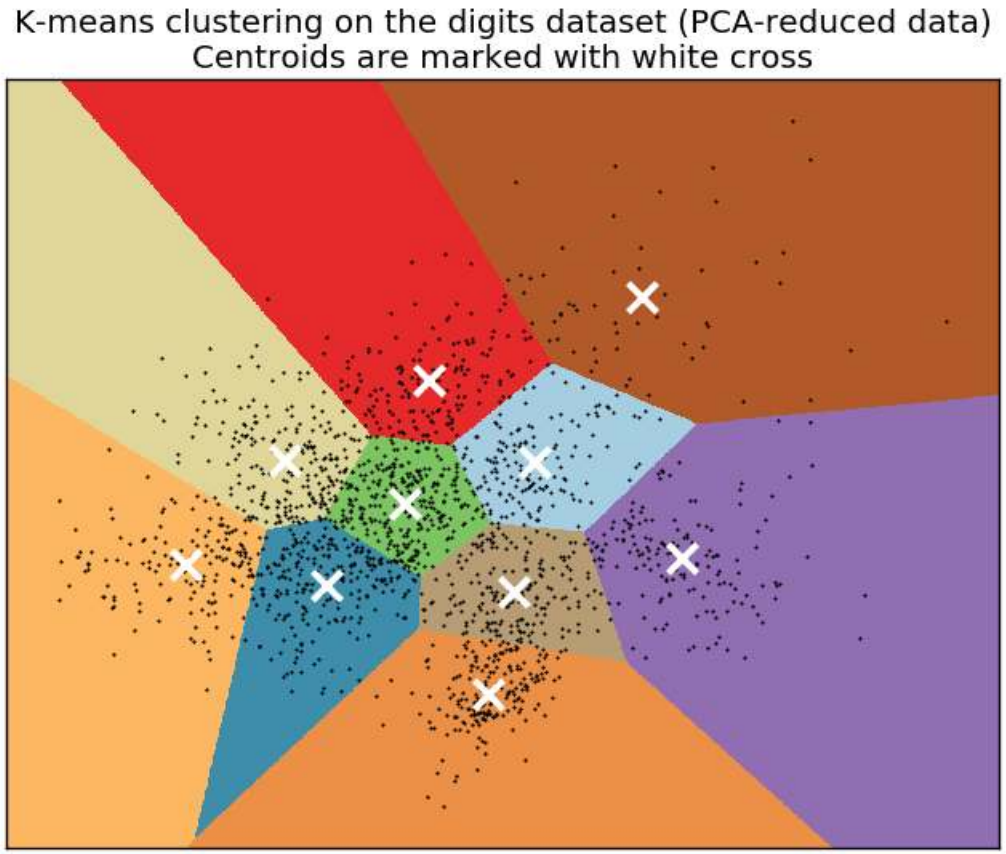
A well-known clustering algorithm: K-Means.

K-Means

- Clusters data by trying to separate the p samples $\mathbf{x}_i$ in K groups of equal variance, minimizing the "*distortion*" $\mathbf{J}(\Phi)$;
- This algorithm requires K, the number of clusters, to be specified before-hand (as most unsupervised models);
- It scales well to large number of samples, and has been used across a large range of application areas in many different fields.

K-Means: a first unsupervised algorithm

A well-known clustering algorithm: K-Means.



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

Description of K-Means

The K-Means algorithm

- Divides a set of p samples $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_p\}$, 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"[a];

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

Description of K-Means

The K-Means algorithm

- Divides a set of p samples $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_p\}$, 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"[a];
- Aims to choose centroids that minimize the distortion, (inertia, or within-cluster sum of squared distances):

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

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

Convergence & implementation

Convergence ?

- 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;
- . . . But it can fall down to a local minimum: that's why a dynamic unsupervised learning algorithm can be useful !

Convergence & implementation

Convergence ?

- 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;
- . . . But it can fall down to a local minimum: that's why a dynamic unsupervised learning algorithm can be useful !

Implementation ?

- K-Means is quick and efficient (with K-Means++ initialization), usually converges, and is easy to implement;
- Available in `scikit-learn`: `sklearn.clustering.KMeans`;
- Also reimplemented myself, see `kmeans.py` (on-line).

Application: color quantization for photos

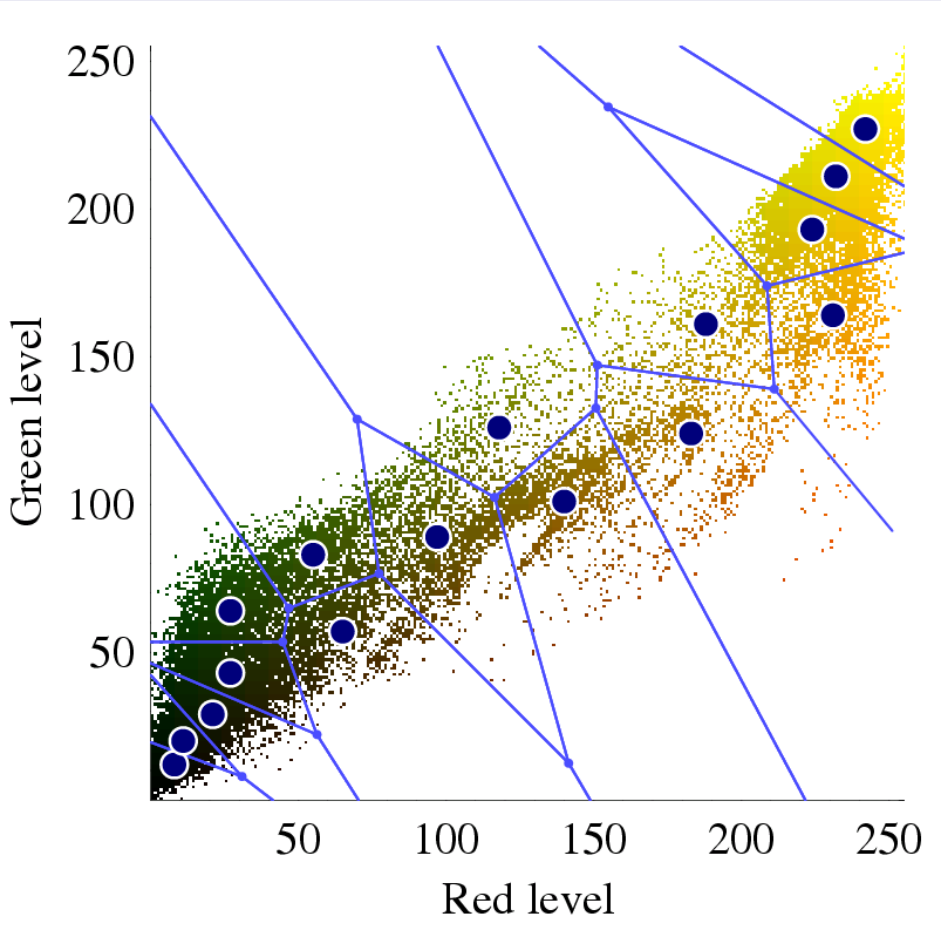
With a two-color-channel image (red/green)

Picture of a flower “Rosa gold glow” (from Wikipedia).

Application: color quantization for photos

In the 2D color-space

Compress the image, by clustering its colors into only 16 **Voronoï diagrams**:



Application: color quantization for photos

"Magnification law"

K-Means fits the **magnification law**:

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

Color quantization for a real-world photo

Color quantization compression on a HD photo:

Heimaey (in Iceland), 3648×2736 pixels, 75986 colors.

Color quantization for a real-world photo

3648 × 2736 pixels, 32 colors from a random codebook.

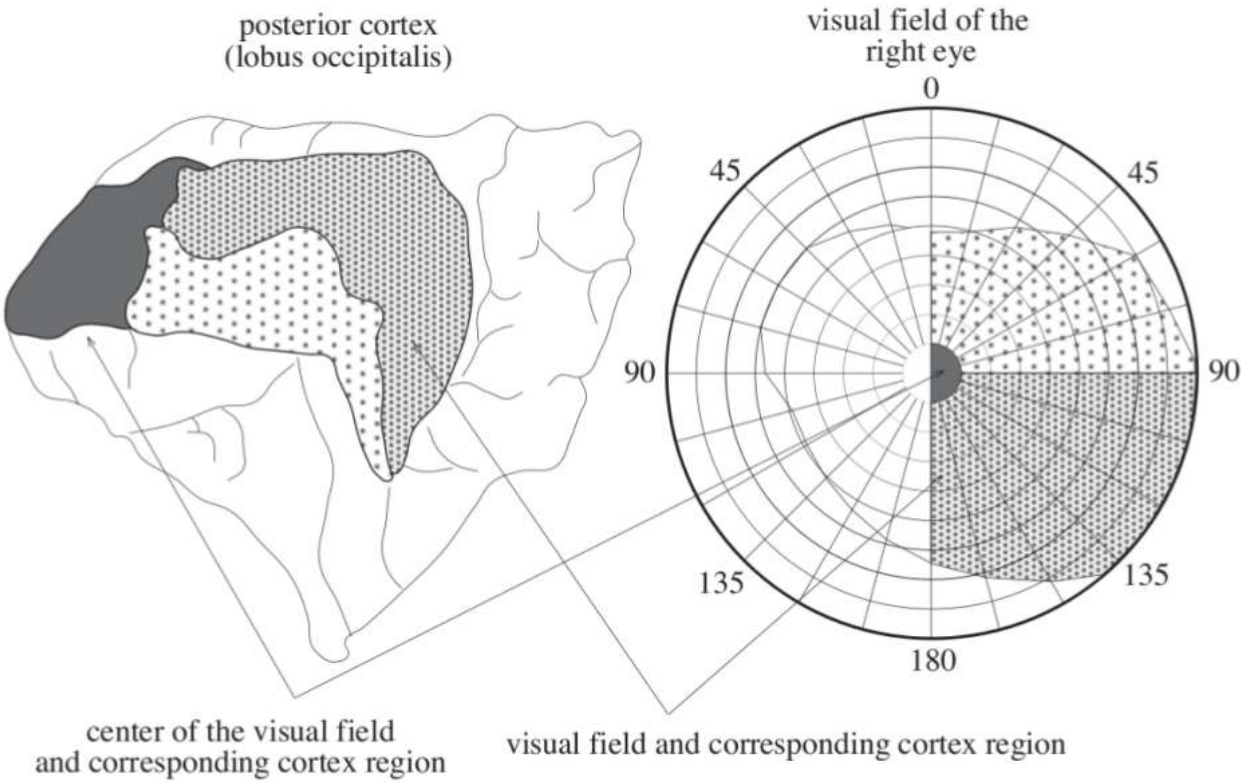
Color quantization for a real-world photo

3648×2736 pixels, 32 colors from a K-Means codebook.

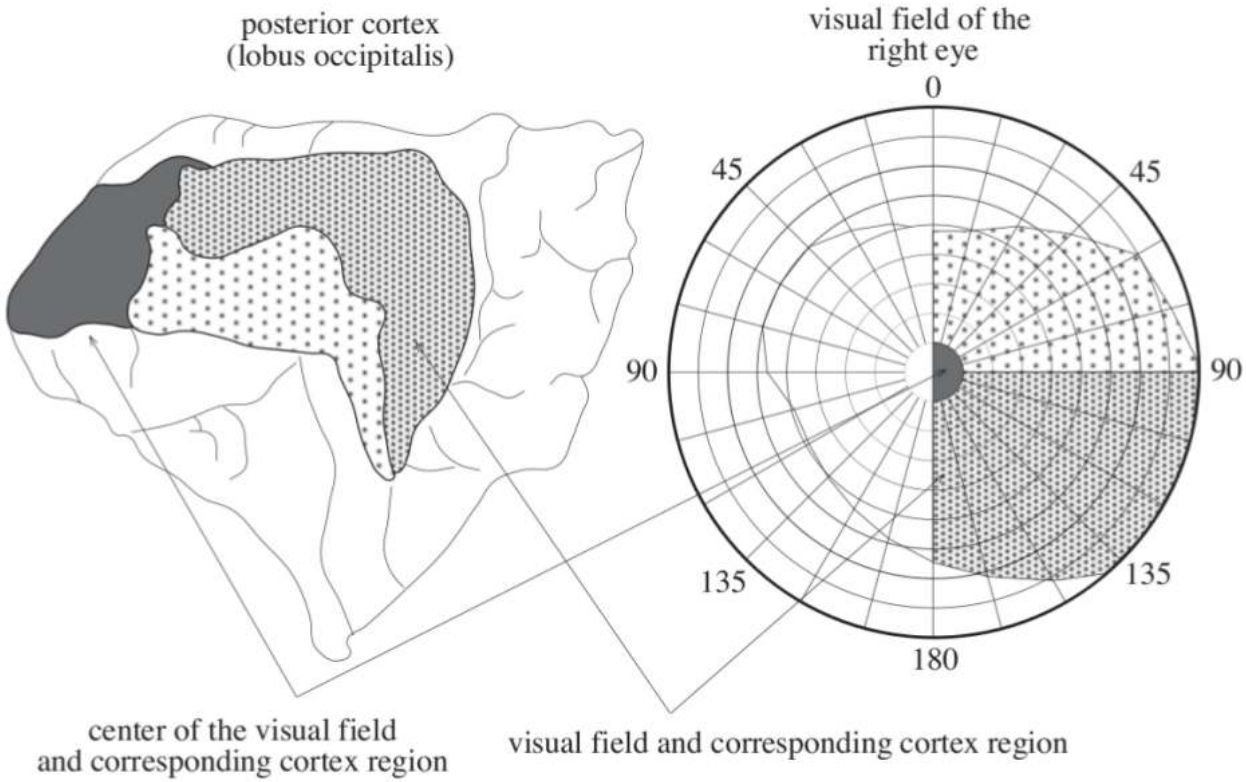
$\Longrightarrow$ (theoretical) compression by a factor $\simeq 2000$: that's **huge**!

A biologically inspired model

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**.



A biologically inspired model



This is referred as "Retinotropic" Organization.

In 1982, from these observations, T. Kohonen tried to model the spatial organization of the visual cortex $\Longrightarrow$ Self-Organizing Map (SOM).

2.1. The SOM model

SOM: how does it work?

– Consider a map of n neurons, **fully** inter-connected;

2.1. The SOM model

SOM: how does it work?

- Consider a map of n neurons, **fully** inter-connected;
- We add a topology on the map, in $\mathbb{R}^q$;

2.1. The SOM model

SOM: how does it work?

- Consider a map of n neurons, **fully** inter-connected;
- We add a topology on the map, in $\mathbb{R}^q$;
- Each neuron i is linked with **all** the input signal (the weight vector $\mathbf{w}_i$ is called the "prototype" of a neuron);

2.1. The SOM model

SOM: how does it work?

- Consider a map of n neurons, **fully** inter-connected;
- We add a topology on the map, in $\mathbb{R}^q$;
- 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;

2.1. The SOM model

SOM: how does it work?

- Consider a map of n neurons, **fully** inter-connected;
- We add a topology on the map, in $\mathbb{R}^q$;
- 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;
- Prototypes of the winner (and his neighbors) are updated, to become closer to the input data.

2.1. The SOM model

SOM: how does it work?

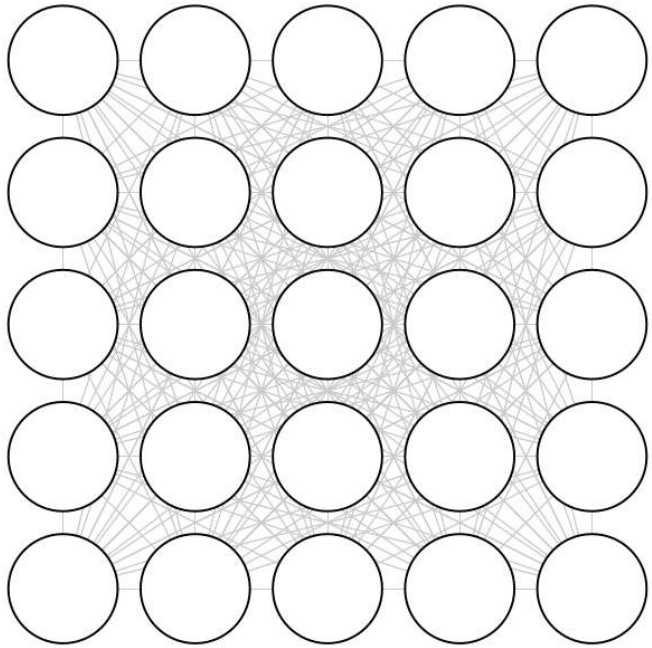
- Consider a map of n neurons, **fully** inter-connected;
- We add a topology on the map, in $\mathbb{R}^q$;
- Each neuron i is linked with **all** the input signal (the weight vector $\mathbf{w}_i$ is called the "prototype" of a neuron);
- Each time a new input data $\mathbf{x}$ is presented, the neuron with the closest prototype wins;
- Prototypes of the winner (and his neighbors) are updated, to become closer to the input data.

And iterate as long as we have training data (or cycle back).

Illustrations: neuronal map

Consider a map of n neurons, fully inter-connected:

so each neuron is linked with any others.

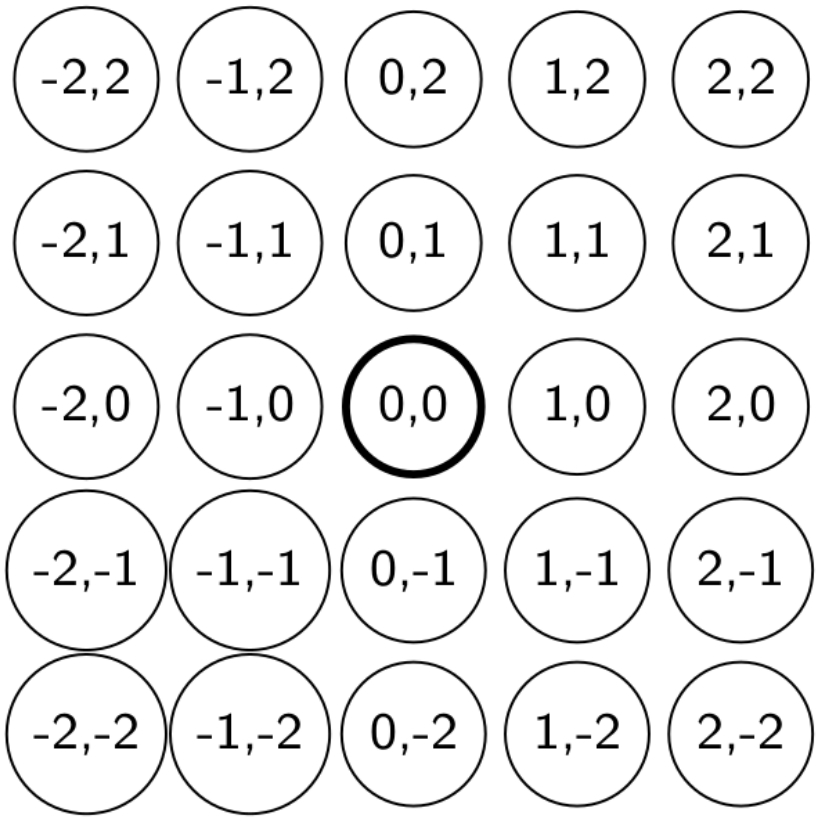


5×5 fully inter-connected neuronal map.

Note: each neuron i has a **fixed** position $\mathbf{p}_i$ in $\mathbb{R}^q$ ($q = 2, 3$ usually).

Illustrations: neuronal map

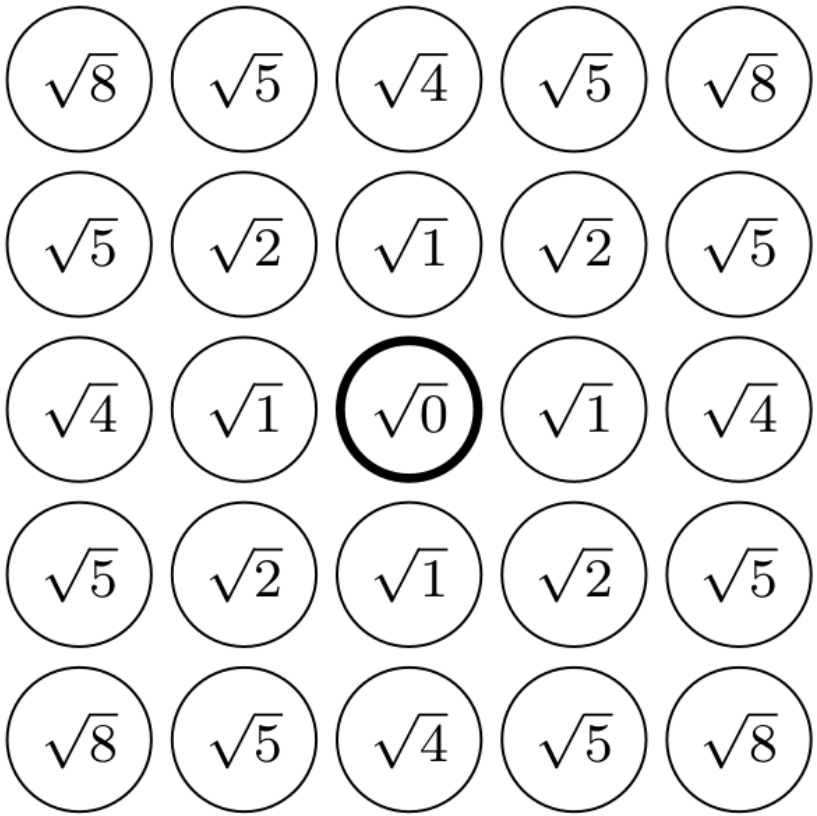
We add a topology on the map, with natural coordinates in $\mathbb{R}^q$.



Coordinates for this 5×5 dense neuronal map.

Illustrations: neuronal map

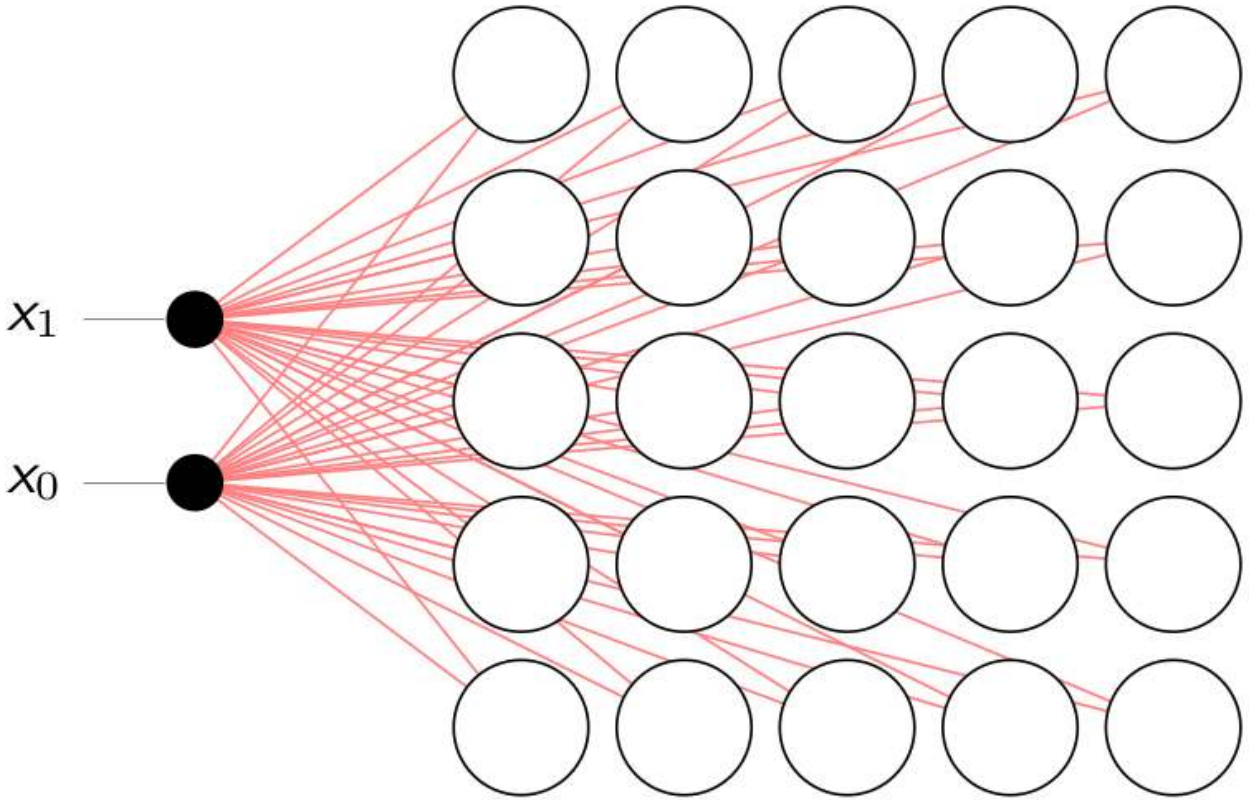
There is an inter-neuron Euclidean distance $\|\cdot\|$.



Euclidean distances for this 5×5 dense neuronal map.

Illustrations: neuronal map

Each neuron is linked with **all** input signals $\mathbf{x}$, the weight vector $\mathbf{w}_i$ is called the "prototype" of a neuron i :



Example of two inputs $\mathbf{x}_0, \mathbf{x}_1$ for this 5×5 dense neuronal map.

SOM learning algorithm: two repeated steps

1. Choosing the winning neuron

Simply $\arg\min$ of the distance between $\mathbf{x}$ (new input) and the prototypes $\mathbf{w}_i$: $i_{\text{win}} \in \arg\min_{i=1..n} d(\mathbf{x}, \mathbf{w}_i)$.

$\Longrightarrow$ *Issue: Need for a centralized entity, not distributed*

(not a very realistic model of cortex organization).

SOM learning algorithm: two repeated steps

1. Choosing the winning neuron

Simply $\arg\min$ of the distance between $\mathbf{x}$ (new input) and the prototypes $\mathbf{w}_i$: $i_{\text{win}} \in \arg\min_{i=1..n} d(\mathbf{x}, \mathbf{w}_i)$.

2. Learning step

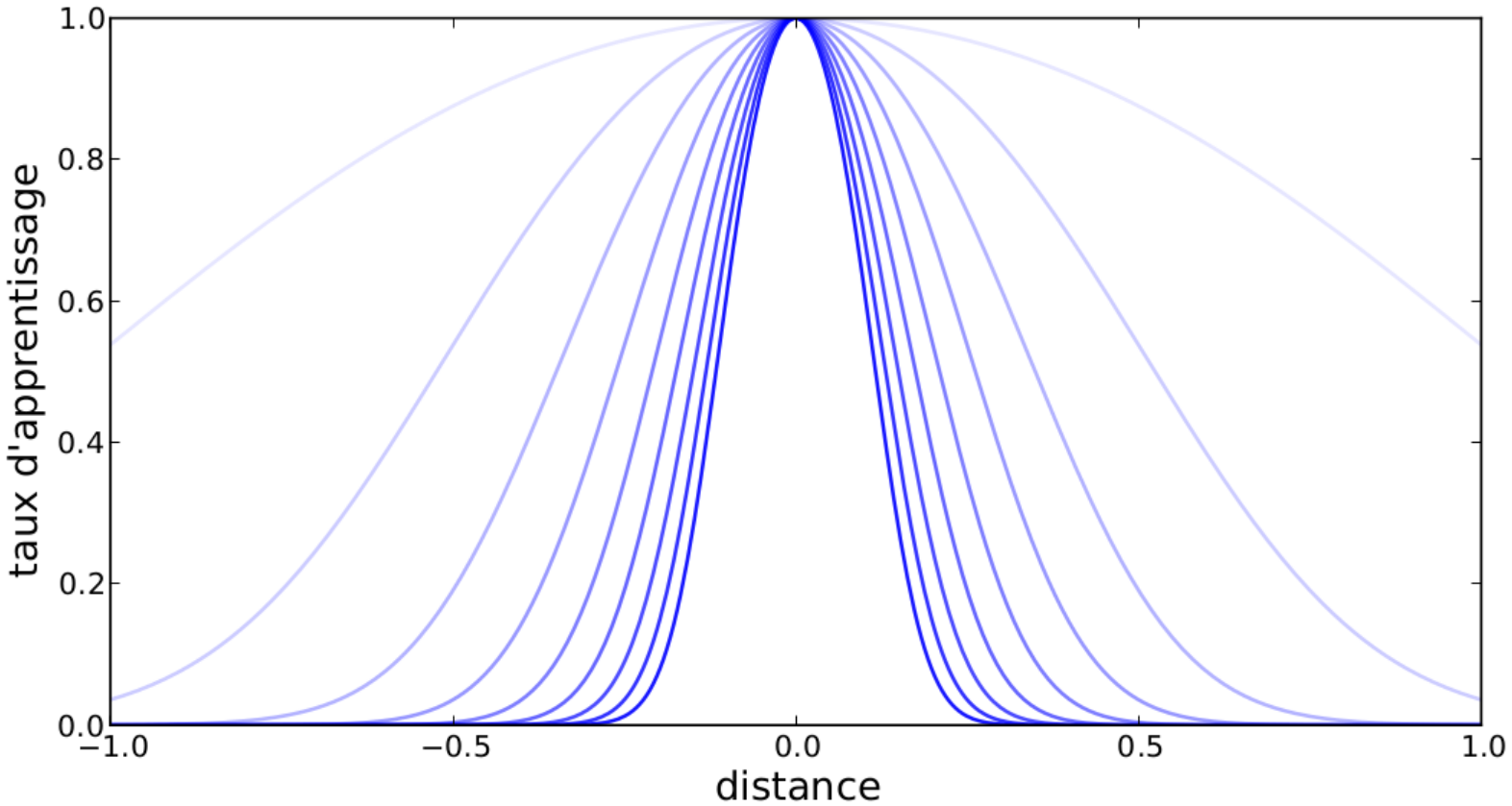
At each new input $\mathbf{x}$, the winning unit (and its neighbors) will update their prototypes with:

$$\mathbf{w}_i(t+1) \leftarrow \mathbf{w}_i(t) + \varepsilon(t) \cdot h(\|\mathbf{p}_i - \mathbf{p}_{i_{\text{win}}}\|) \cdot (\mathbf{w}_i(t) - \mathbf{x})$$

- $\varepsilon(t) > 0$ is a (decreasing) learning rate;
- $h(\cdot)$ is a neighborhood function, on distances between neurons ($\|\mathbf{p}_i - \mathbf{p}_{i_{\text{win}}}\|$).

Neighborhood on the neuronal map

The neighborhood function *only* depends on the distance of $\mathbf{p}_i$ from the winning neuron: (fully isotropic model)



Neighborhood function of **distance from the winning neuron** ($\|\mathbf{p}_i - \mathbf{p}_{i_{\text{win}}}\|$).

Parameters and specification of a SOM

Learning time $t = t_{\text{init}} \dots t_{\text{end}}$

Starting at $t_{\text{init}} = 0$, and finishing at $t_{\text{end}} = t_f \in \mathbb{N}^*$.

$\Longrightarrow$ Issue: t_f has to be decided in advanced.

Parameters and specification of a SOM

Vectorial update rule: $\Delta \mathbf{w}_i \overset{\text{def}}{=} \varepsilon(t) \cdot h_\sigma(t, i, i_{\text{win}}) \cdot (\mathbf{x} - \mathbf{w}_i)$.

Learning rate $\varepsilon(t)$

$\varepsilon(t)$ is a (geometrically) decreasing **learning rate**.
We choose $0 \leqslant \varepsilon_{\text{end}} \ll \varepsilon_{\text{init}}$:

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

$\Longrightarrow$ Issue: the map is (almost) fixed after a certain time (not online learning, not dynamic).

Parameters and specification of a SOM

Vectorial update rule: $\Delta \mathbf{w}_i \stackrel{\text{def}}{=} \varepsilon(t) \cdot h_\sigma(t, i, i_{\text{win}}) \cdot (\mathbf{x} - \mathbf{w}_i)$.

Neighborhood function h_σ and width $\sigma(t)$

$h_\sigma(t, i, i_{\text{win}})$ is a neighborhood function, usual form is a Gaussian:

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

$\sigma(t)$ is a (geometrically) decreasing **width**.
We choose $0 < \sigma_{\text{end}} \ll \sigma_{\text{init}}$:

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

2.2. The Neural Gas model

Very similar to a SOM, but no underlying topology for the neuron space $\mathbb{R}^q$. Just prototypes $\mathbf{w}_i$.

2.2. The Neural Gas model

Very similar to a SOM, but no underlying topology for the neuron space $\mathbb{R}^q$. Just prototypes $\mathbf{w}_i$.
For a new input $\mathbf{x}$, all neurons i are **ordered by increasing distance** of $\mathbf{w}_i$ to $\mathbf{x}$, and assigned a **rank** $k_i(\mathbf{x})$ (in $[1..n]$).

2.2. The Neural Gas model

Very similar to a SOM, but no underlying topology for the neuron space $\mathbb{R}^q$. Just prototypes $\mathbf{w}_i$.
For a new input $\mathbf{x}$, all neurons i are **ordered by increasing distance** of $\mathbf{w}_i$ to $\mathbf{x}$, and assigned a **rank** $k_i(\mathbf{x})$ (in $[1..n]$).
The update rule is modified to be:

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

- Same learning rate $\varepsilon(t)$ and width $\sigma(t)$, decreasing with time.
- But the neighborhood function is now a inverse exponential on **ranks**: $h_\sigma(t, i, \mathbf{x}) \stackrel{\text{def}}{=} \exp\left(-k_i(\mathbf{x})/\sigma(t)\right)$.

2.2. The Neural Gas model

Very similar to a SOM, but no underlying topology for the neuron space $\mathbb{R}^q$. Just prototypes $\mathbf{w}_i$.
For a new input $\mathbf{x}$, all neurons i are **ordered by increasing distance** of $\mathbf{w}_i$ to $\mathbf{x}$, and assigned a **rank** $k_i(\mathbf{x})$ (in $[1..n]$).
The update rule is modified to be:

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

- Same learning rate $\varepsilon(t)$ and width $\sigma(t)$, decreasing with time.
- But the neighborhood function is now a inverse exponential on **ranks**: $h_\sigma(t, i, \mathbf{x}) \overset{\text{def}}{=} \exp\left(-k_i(\mathbf{x})/\sigma(t)\right)$.

Not covered more: don't have time.

Cf. [Rougier and Boniface, 2011a].

Extensions: Growing or Dynamic Neural Gas

Online learning with Neural Gas ?

There is also some extensions to the Neural Gas model: Growing NG or Dynamic NG.

But "not today" . . .

I have not studied these extensions.

2.3. The Neural Fields model

Dynamic **Neural Fields**: another family of model, inspired from the continuous LeapField model (from MEEG), rather than neural networks.

2.3. The Neural Fields model

Dynamic **Neural Fields**: another family of model, inspired from the continuous LeapField model (from MEEG), rather than neural networks.

They consider a continuous membrane potential, following a functional PDE:

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

2.3. The Neural Fields model

Dynamic **Neural Fields**: another family of model, inspired from the continuous LeapField model (from MEEG), rather than neural networks.
They consider a continuous membrane potential, following a functional PDE:

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

- $V(\mathbf{x},t)$ is the membrane potential at position $\mathbf{x}$ and time t;
- $W(\|\mathbf{x}-\mathbf{y}\|)$ is the lateral connection weight between $\mathbf{x}$ and $\mathbf{y}$;
- f is the mean firing rate, and h is the resting potential;
- $I(\mathbf{x},t)$ is the input at position $\mathbf{x}$.

2.3. The Neural Fields model

Dynamic **Neural Fields**: another family of model, inspired from the continuous LeapField model (from MEEG), rather than neural networks.
They consider a continuous membrane potential, following a functional PDE:

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

- $V(\mathbf{x},t)$ is the membrane potential at position $\mathbf{x}$ and time t;
- $W(\|\mathbf{x}-\mathbf{y}\|)$ is the lateral connection weight between $\mathbf{x}$ and $\mathbf{y}$;
- f is the mean firing rate, and h is the resting potential;
- $I(\mathbf{x},t)$ is the input at position $\mathbf{x}$.

The PDE is solved with a numerical discretization $(V(\mathbf{x}_i,t), i = 1..n, t = t_{\text{init}}..t_{\text{end}},)$, and a forward Euler scheme.
Not covered more: don't have time.

Cf. [Rougier and Detorakis, 2011].

Extension: Self-Organizing DNF

In 2011, N. Rougier and Y. Boniface introduced an extension of the DNF model to model self-organization with a Neural Field.

Modified learning rule

- 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.

(Simple relaxation of the previously used learning rate.)

Not covered more: don't have time.

Cf. [Rougier and Detorakis, 2011].

Back to the SOM model

Back to the Self-Organizing Map (SOM) model.

The SOM model has some weaknesses

A few issues:

The SOM model has some weaknesses

A few issues:

- The map topology can not correspond to the data topology, this can ruins the learning possibility;

The SOM model has some weaknesses

A few issues:

- The map topology can not correspond to the data topology, this can ruins the learning possibility;
- The map can fail to deploy correctly in the first learning steps, and we get big aggregates of prototypes ($\Longrightarrow$ local minimum of distortion);

The SOM model has some weaknesses

A few issues:

- The map topology can not correspond to the data topology, this can ruins the learning possibility;
- The map can fail to deploy correctly in the first learning steps, and we get big aggregates of prototypes
  ($\Longrightarrow$ local minimum of distortion);
- The map is fixed after training, as learning rate goes to $\varepsilon_{\text{end}} \ll 1$ (no long-term learning, only stationary distributions).
  $\Longrightarrow$ models part of the learning process in early years;

The SOM model has some weaknesses

A few issues:

- The map topology can not correspond to the data topology, this can ruins the learning possibility;
- The map can fail to deploy correctly in the first learning steps, and we get big aggregates of prototypes
  ($\Longrightarrow$ local minimum of distortion);
- The map is fixed after training, as learning rate goes to $\varepsilon_{\text{end}} \ll 1$ (no long-term learning, only stationary distributions).
  $\Longrightarrow$ models part of the learning process in early years;
- We have to know the ending learning time t_f in advance, i.e. number of training examples given to the map (no online learning).

Constant learning rate on a SOM $\Longrightarrow$ DSOM

Simply 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):

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

– $\varepsilon_0 > 0$ is the constant learning rate;

Constant learning rate on a SOM $\Longrightarrow$ DSOM

Simply 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):

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

- $\varepsilon_0 > 0$ is the constant learning rate;
- $\eta > 0$ is the elasticity / plasticity parameter;

Constant learning rate on a SOM $\Longrightarrow$ DSOM

Simply 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):

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

- $\varepsilon_0 > 0$ is the constant learning rate;
- $\eta > 0$ is the elasticity / plasticity parameter;
- h_η is a time-invariant neighborhood[1] function:

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

It is like having time-invariant but *local dependent* learning rate $\varepsilon(t)$ & width $\sigma(t)$.

[1] Convention: $h_\eta(i, i_{\text{win}}, \mathbf{x}) \overset{\text{def}}{=} 0$ if $\mathbf{x} = \mathbf{w}_{i_{\text{win}}}$.

Consequences of a constant learning rate

1. Online learning

No need for end time t_f : can accept data as long as needed.

Consequences of a constant learning rate

1. Online learning

No need for end time t_f : can accept data as long as needed.

2. 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.

Consequences of a constant learning rate

1. Online learning

No need for end time t_f : can accept data as long as needed.

2. 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.

3. Different parameters (less parameters !)

Instead of 5 parameters t_f, σ_{init}, σ_{end}, $\varepsilon_{\text{init}}$, ε_{end}; only need for 2: constant learning rate ε_0 and an elasticity η.

Consequences of a constant learning rate

1. Online learning

No need for end time t_f : can accept data as long as needed.

2. 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.

3. Different parameters (less parameters !)

Instead of 5 parameters t_f, σ_{init}, σ_{end}, $\varepsilon_{\text{init}}$, ε_{end}; only need for 2: constant learning rate ε_0 and an elasticity η.

But . . .

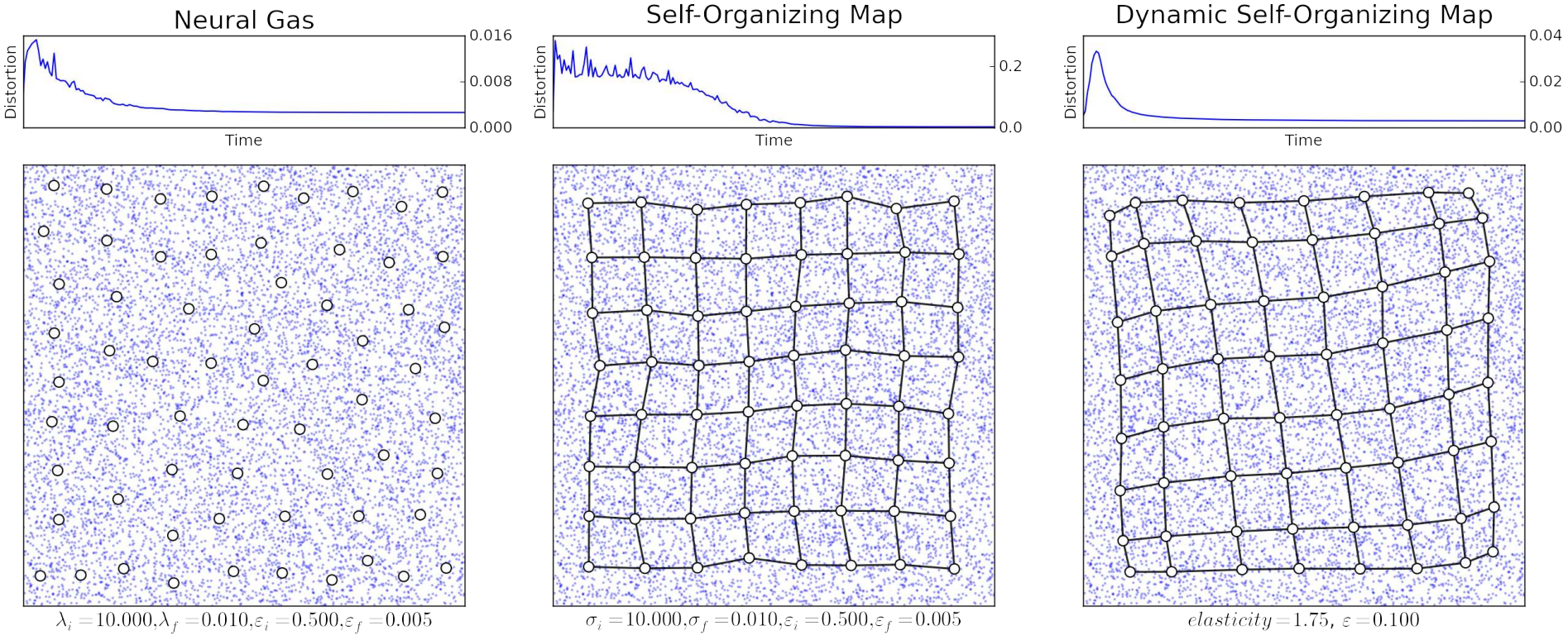
But convergence seems harder, and stability is not achievable: less theoretical guarantee.

Comparisons between NG, SOM and DSOM

Experimental setup (Experiments 1/2)

- 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 $[0, 1] \times [0, 1]$.
- 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 distribution / map.
- Small blue points are the training samples $\mathbf{x}_j$, big **white points** are the vectors of the codebook $\mathbf{w}_i$.

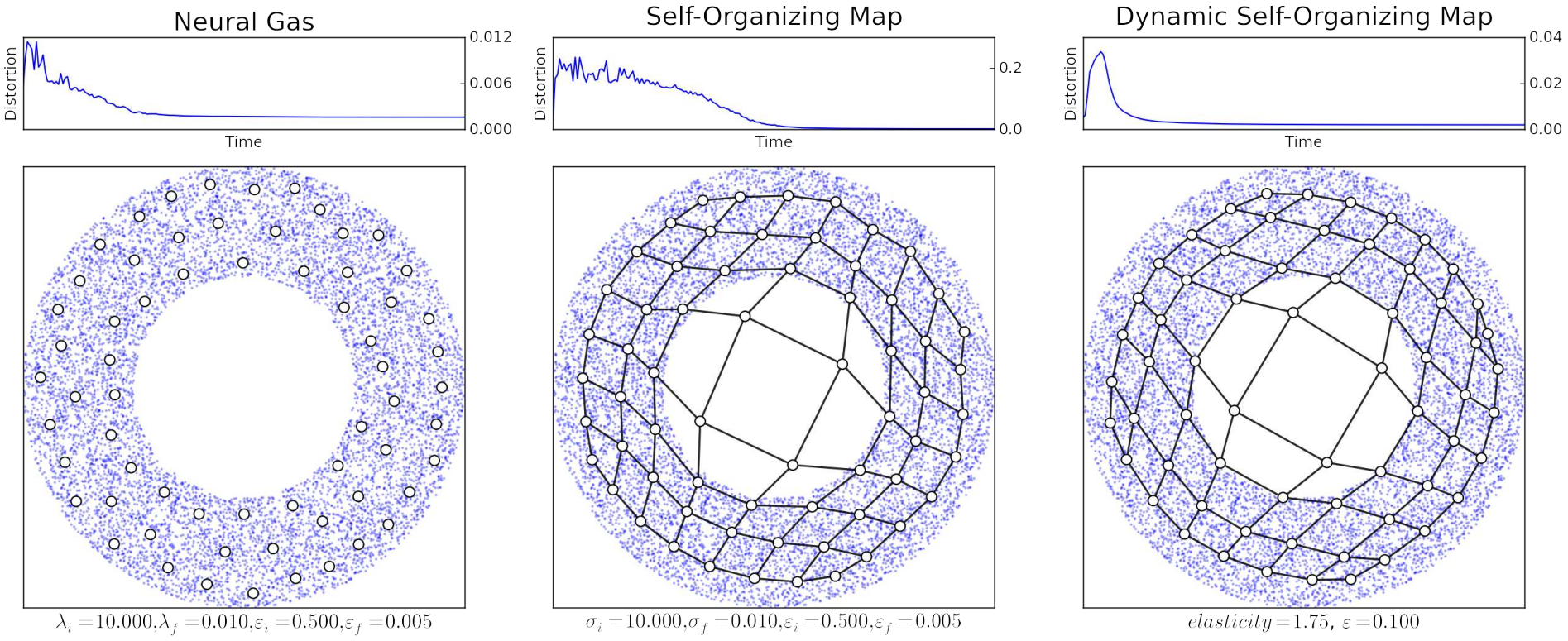
Comparisons between NG, SOM and DSOM



A simple uniform distribution.

– DSOM gives a smoother map than SOM.

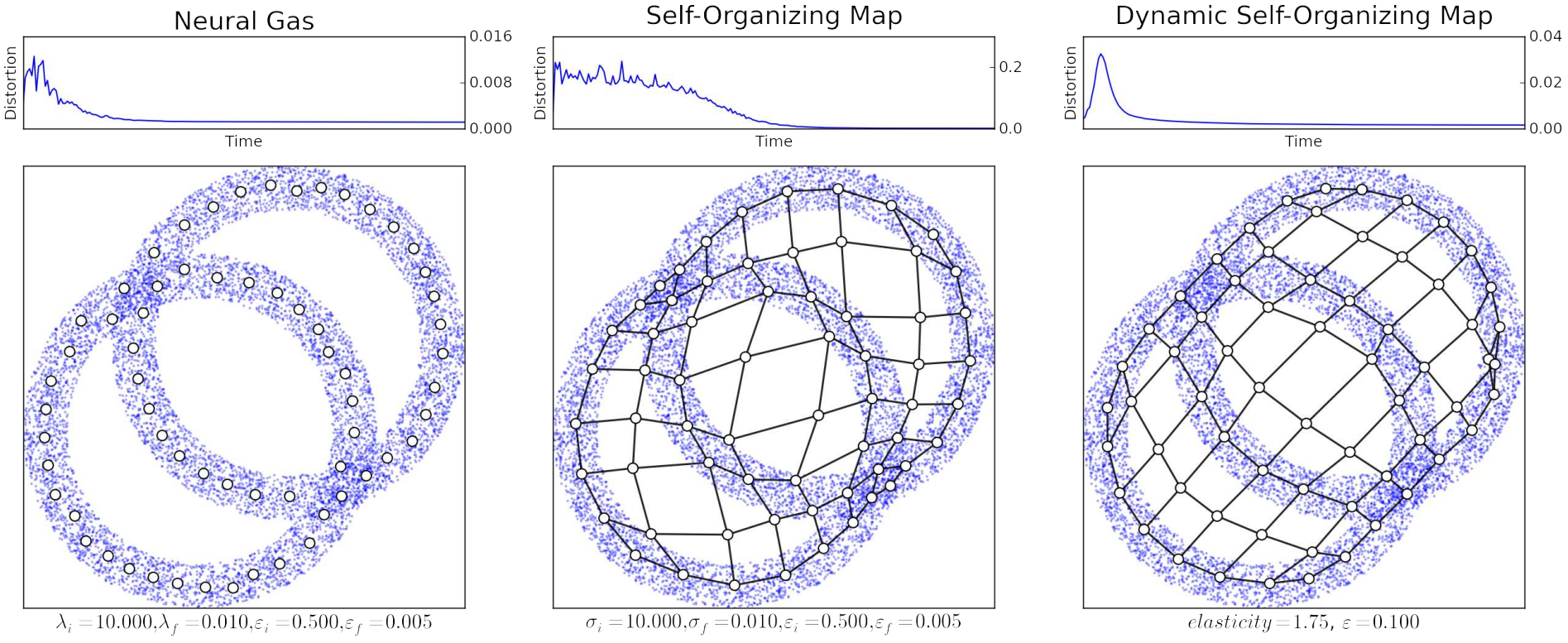
Comparisons between NG, SOM and DSOM



A simple ring distribution.

– Distortion decreases more quickly/smoothly with DSOM than a NG/SOM.

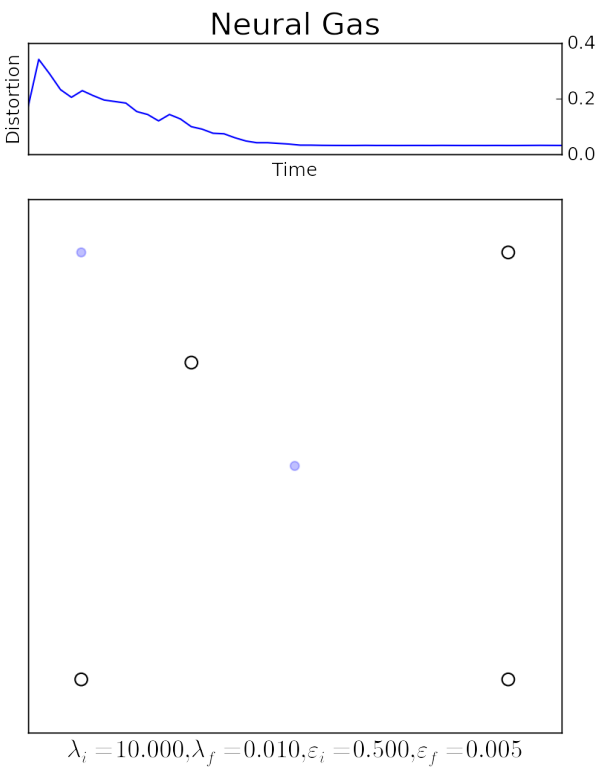
Comparisons between NG, SOM and DSOM

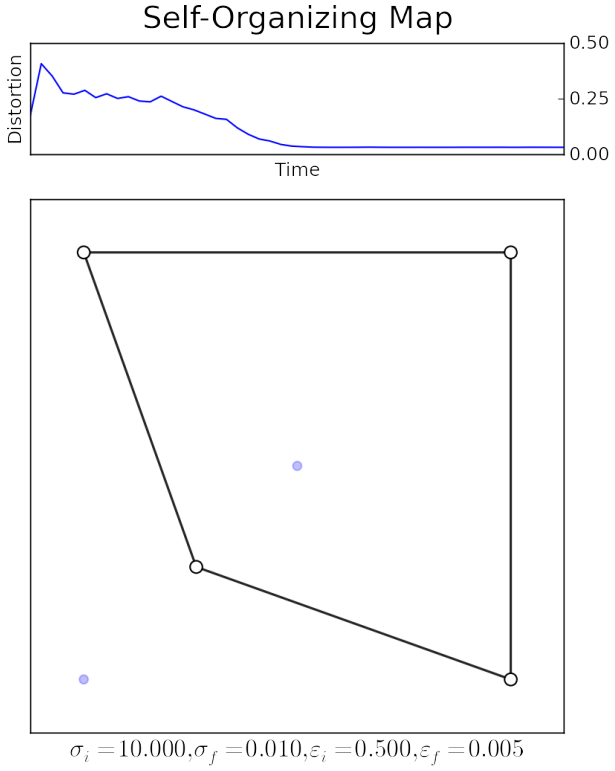


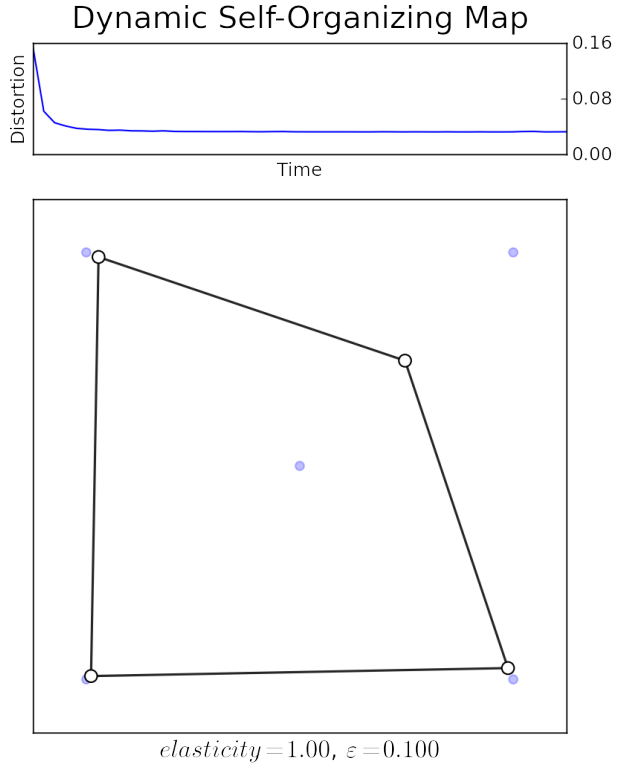
Double ring distribution.

– NG achieves here a lower distortion (SOM/DSOM have useless nodes).

Comparisons between NG, SOM and DSOM



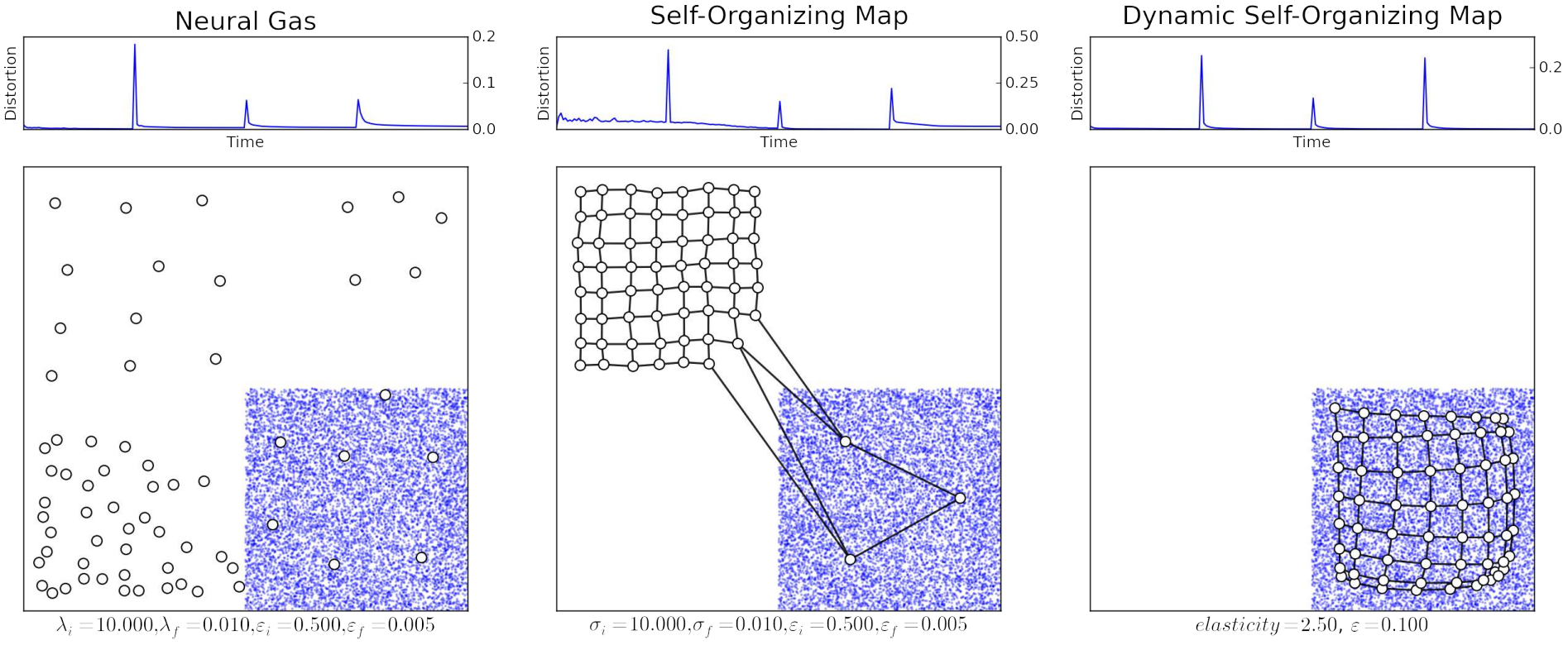




Issue for wrongly designed topology: 4 nodes for 5 data points.

– SOM/DSOM are not great here.

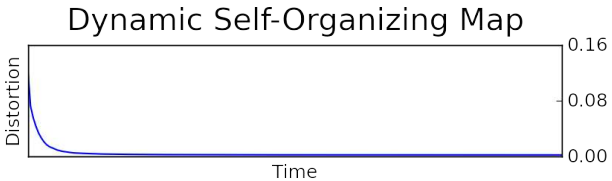
Comparisons between NG, SOM and DSOM

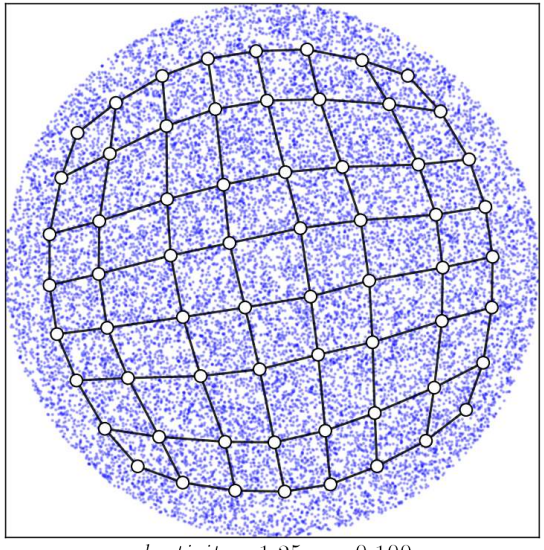


Non-stationary distribution, moving from quarters $3 \rightarrow 2 \rightarrow 1 \rightarrow 4$.

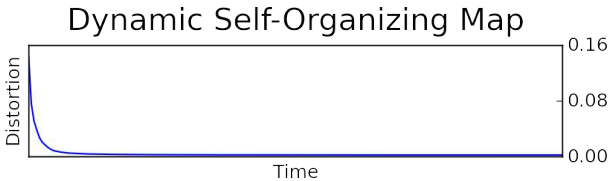
– DSOM allows **long-term learning**: model cortical plasticity as a tight coupling between model and environment.

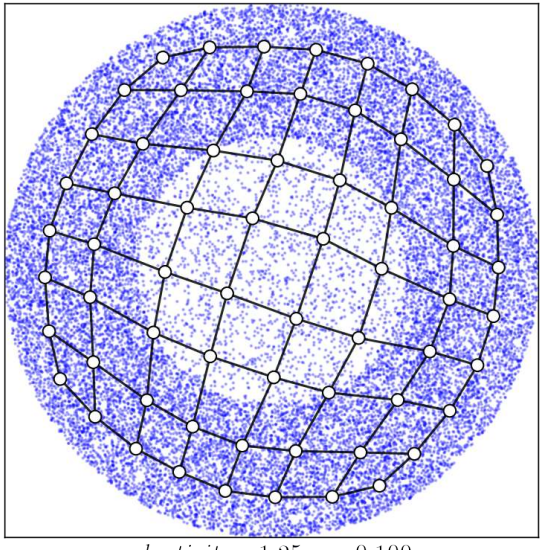
Magnification law for a DSOM ?



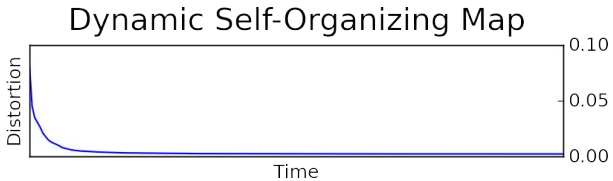


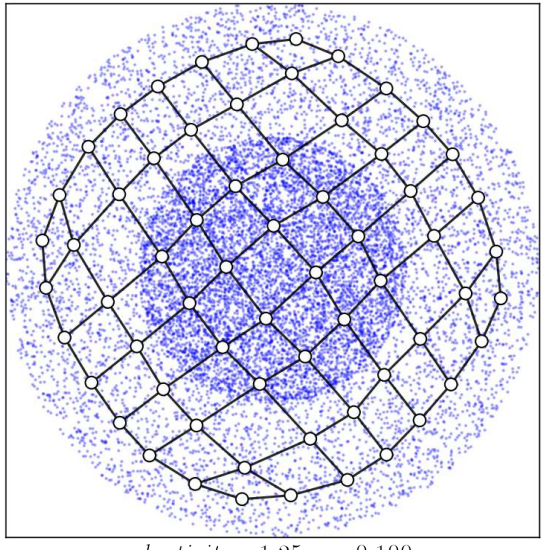
$elasticity = 1.25,\ \varepsilon = 0.100$





$elasticity = 1.25,\ \varepsilon = 0.100$





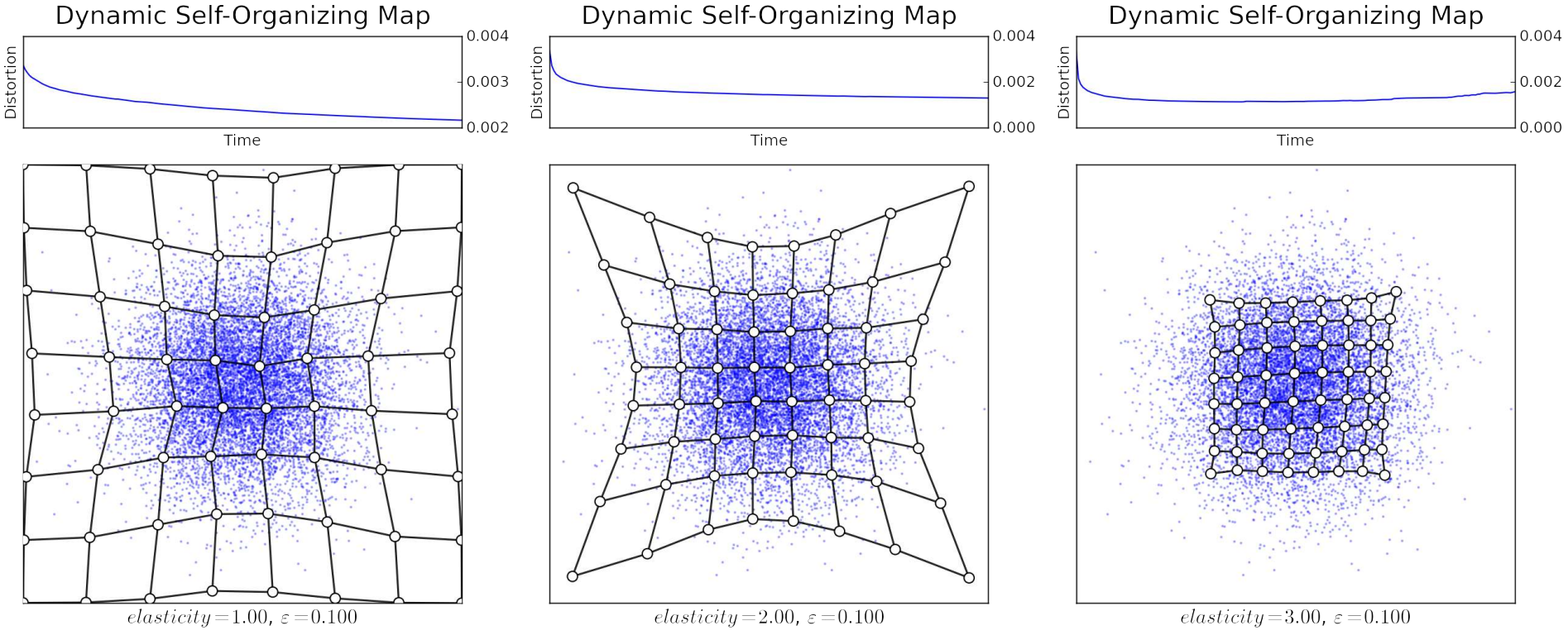
$elasticity = 1.25,\ \varepsilon = 0.100$

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

$\Longrightarrow$ DSOM does not fits the "magnification law".

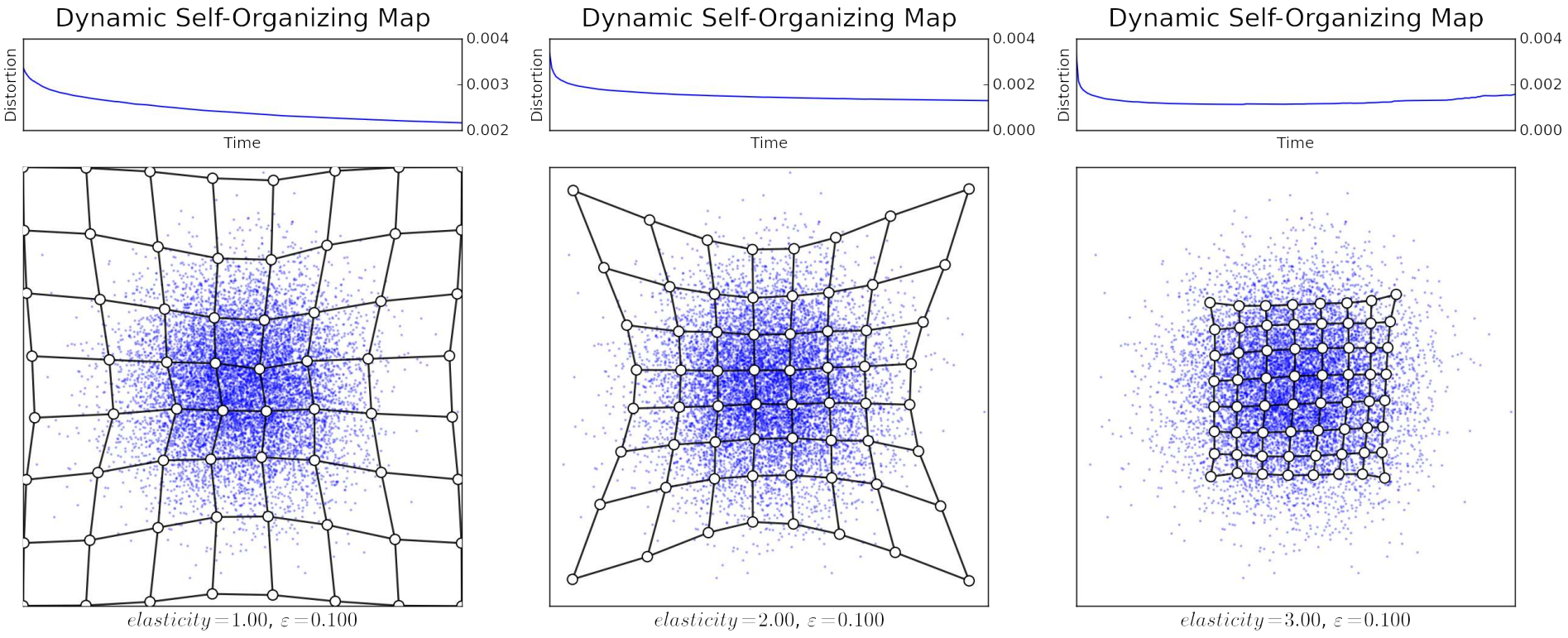
Is it good news or bad news? Depend on the application.

Influence of the elasticity parameter



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

Influence of the elasticity parameter



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

Can we find a way to auto-tune the elasticity or width parameter? σ_{init} and σ_{end} for SOM, and η for DSOM.
Probably not . . . A grid search for both, based on distortion, cannot do the job.

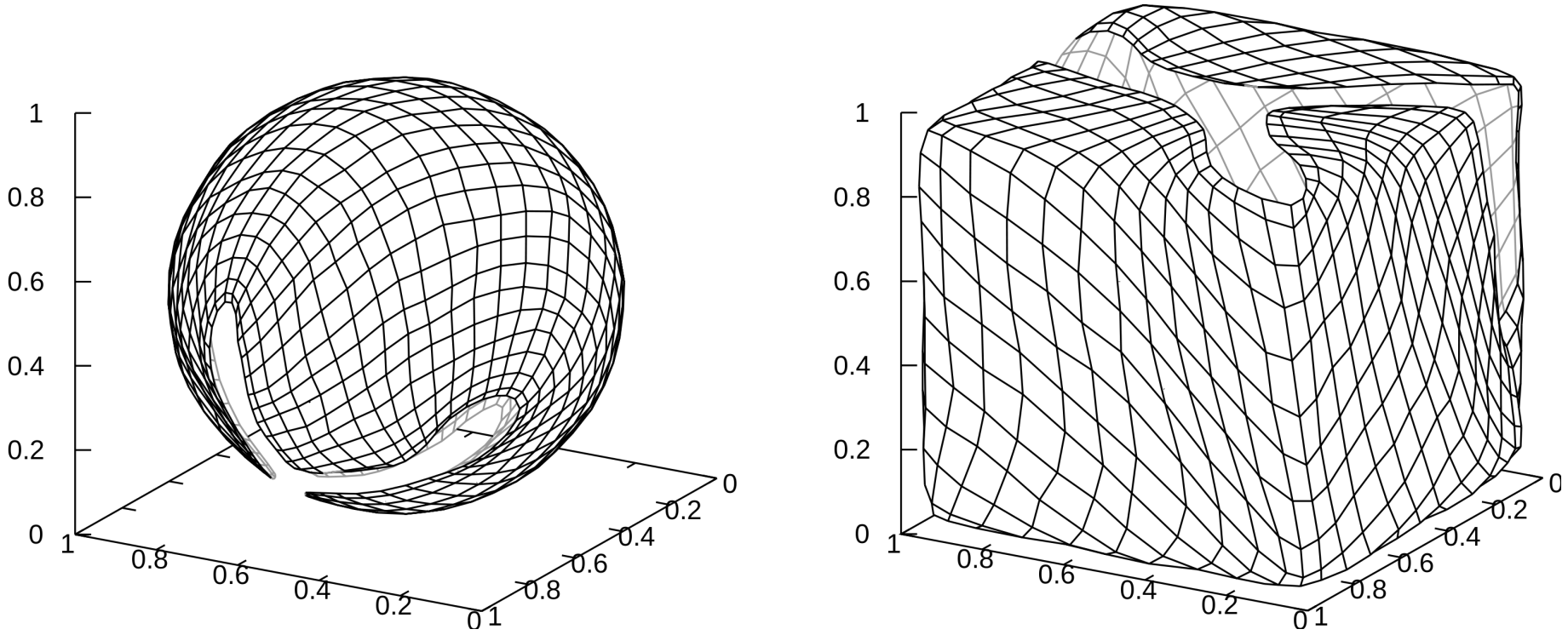
Examples of non-stationary distributions

Experimental setup (Experiments 2/2)

- 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;
- And elasticity 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).

Examples of non-stationary distributions

Another example of non-stationary distribution, a $2D$ manifold continuously changed from a sphere to a cube (in $3D$). *Cf. animations.*



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

A few harder questions

What if $d \geqslant 2, 3$?

What topology to adopt for higher dimension data?
Example of image processing with NG/SOM/DSOM in [Rougier and Boniface, 2011a]: vectorial quantization on a similarity graph from small patches of an image.

A few harder questions

What if $d \geqslant 2, 3$?

What topology to adopt for higher dimension data?
Example of image processing with NG/SOM/DSOM in [Rougier and Boniface, 2011a]: vectorial quantization on a similarity graph from small patches of an image.

Separate distributions ?

If there is a need for a topological rupture: how to let a DSOM decides to split in 2 (or more) sub-maps?

A few harder questions

What if $d \geqslant 2, 3$?

What topology to adopt for higher dimension data?
Example of image processing with NG/SOM/DSOM in [Rougier and Boniface, 2011a]: vectorial quantization on a similarity graph from small patches of an image.

Separate distributions ?

If there is a need for a topological rupture: how to let a DSOM decides to split in 2 (or more) sub-maps?

Theoretical warranties ?

Convergence and stability: not proved.
Stability even seems unachievable if we want to keep long-term learning (online learning).

Quick sum-up . . . I

We recalled . . .

- Different types of learning; (in the brain and in machine learning)
- Unsupervised learning is harder, but it's the future;
- Clustering algorithms are useful, e.g. for data compression and also modeling brain self-organization property.

Quick sum-up . . . II

In particular, we saw . . .

- Several clustering algorithms:
  - K-Means;
  - Neural Gas; (quickly)
  - NF & DNF; (quickly)
  - SOM & DSOM . . .
- Why a dynamic model can be useful.
- Some theoretical and practical questions are still to be answered:
  - automatically choosing elasticity η ?
  - convergence?
  - stability?
  - etc.

Quick sum-up . . . III

Experimentally, we applied . . .

- K-Means and a SOM to color quantization (image compression); [Bloomberg, 2008]
- NG, SOM and DSOM on several stationary and non-stationary distributions in 2D; [Rougier and Boniface, 2011a]
- SOM and DSOM on a higher dimension distribution (from image processing); [Rougier and Boniface, 2011a]

And all experiments confirmed the intuitions about the models.

Thank you!

Thank you for your attention.

. . . and thanks for the course !

Questions ?

Questions ?

Want to know more?

↪ Explore the references, or read the project report (about 15 pages);

↪ And e-mail me if needed: lilian.besson@ens-cachan.fr

Main references

- T. Kohonen (1998), *"The Self-Organizing Map"*, reference book [Kohonen, 1998].
- N.P. Rougier & Y. Boniface (2011), *"Dynamic Self-Organizing Map"*, research article [Rougier and Boniface, 2011a], and code [Rougier and Boniface, 2011b].
- N.P. Rougier, and G. Detorakis (2011), *"Self-Organizing Dynamic Neural Fields"*, research article [Rougier and Detorakis, 2011].

Appendix

Outline of the appendix

- More references given below.
- Code, figures and raw results from some experiments:

  ⟶ http://lbo.k.vu/neuro2016
- Everything here is open-source under the MIT License.

More references . . . I

Main reference

The main reference is the work of N.P. Rougier and Y. Boniface, in 2011, presented in *"Dynamic Self-Organizing Map"* [Rougier and Boniface, 2011a, Rougier and Boniface, 2011b].

More references . . . II


Cottrell, M., Fort, J.-C., and Pagès, G. (1998).
Theoretical Aspects of the SOM Algorithm.
Neurocomputing, 21(1):119–138.


Deng, J. D. and Kasabov, N. K. (2003).
On-line Pattern Analysis by Evolving Self-Organizing Maps.
Neurocomputing, 51:87–103.


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

More references . . . III


Fausett, L. (1994).
Fundamentals of Neural Networks: Architectures, Algorithms, and Applications.
Prentice-Hall, Inc., Upper Saddle River, NJ, USA.


Kohonen, T. (1998).
The Self-Organizing Map.
Neurocomputing, 21(1):1–6.


Rougier, N. P. and Boniface, Y. (2011a).
Dynamic Self-Organizing Map.
Neurocomputing, 74(11):1840–1847.


Rougier, N. P. and Boniface, Y. (2011b).
Dynamic Self-Organizing Map.
Python code sources.

More references . . . IV


Rougier, N. P. and Detorakis, G. (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.


Sutton, R. S. and Barto, A. G. (1998).
Reinforcement Learning: An Introduction, volume 1.
MIT Press, Cambridge, MA.

Open-Source Licensed

License?

These slides and the report[a] are open-sourced under the terms of the **MIT License** (see `lbesson.mit-license.org`).

[a] And the additional resources – including code, figures, etc.