

Artificial Intelligence

Online learning and self-organizing systems

Marco Piastra

Moving averages

An aside: following non-stationary phenomena

■ Average

Definition:
$$\bar{v}_T := \frac{1}{T} \sum_{k=1}^T v_k$$

Running implementation:

$$\begin{aligned}\bar{v}_T &= \frac{1}{T} \left(v_T + \sum_{k=1}^{T-1} v_k \right) = \frac{1}{T} \left(v_T + (T-1) \bar{v}_{T-1} \right) \\ &= \bar{v}_{T-1} + \frac{1}{T} (v_T - \bar{v}_{T-1}) = \frac{1}{T} v_T + \left(1 - \frac{1}{T} \right) \bar{v}_{T-1}\end{aligned}$$

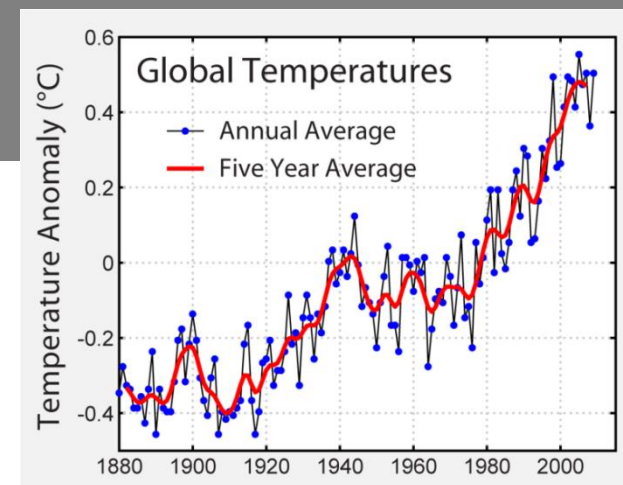
■ Simple Moving Average (SMA)

$$\bar{v}_{T,n} := \frac{1}{n} \sum_{k=T-n}^T v_k$$

■ Exponential Moving Average (EMA)

$$\bar{v}_{T,\alpha} = \alpha v_T + (1 - \alpha) \bar{v}_{T-1,\alpha}, \quad \alpha \in [0, 1]$$

"the weight of newer observations remains constant"



[image from wikipedia]

"the weight of newer observations diminishes with time"

Moving averages

■ Exponential Moving Average (EMA)

$$\bar{v}_{T,\alpha} = \alpha v_T + (1 - \alpha) \bar{v}_{T-1,\alpha}, \quad \alpha \in [0, 1]$$

Expanding:

$$\begin{aligned} \bar{v}_{t,\alpha} &= \alpha v_t + (1 - \alpha) \bar{v}_{t-1,\alpha} \\ &= \alpha v_t + (1 - \alpha)(\alpha v_{t-1} + (1 - \alpha) \bar{v}_{t-2,\alpha}) \\ &= \alpha v_t + (1 - \alpha)(\alpha v_{t-1} + (1 - \alpha)(\alpha v_{t-2} + (1 - \alpha) \bar{v}_{t-3,\alpha})) \\ &= \alpha (v_t + (1 - \alpha) v_{t-1} + (1 - \alpha)^2 v_{t-2}) + (1 - \alpha)^3 \bar{v}_{t-3,\alpha} \end{aligned}$$

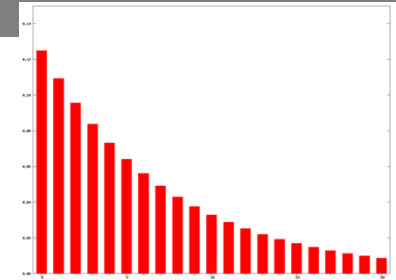
The weight of past contributions decays as

$$(1 - \alpha)^{\Delta t}$$

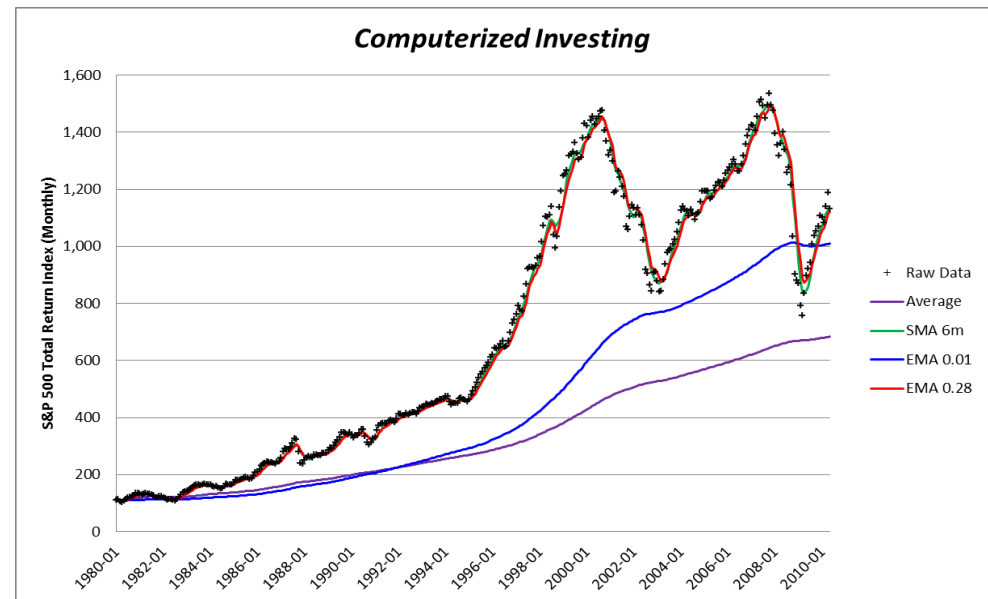
A SMA with n previous values is approximately equal to an EMA with

$$\alpha = \frac{2}{n + 1}$$

$(1 - \alpha)^{\Delta t}$
"the weight
of older observations
diminishes with time"



[image from wikipedia]



Stochastic Gradient Descent

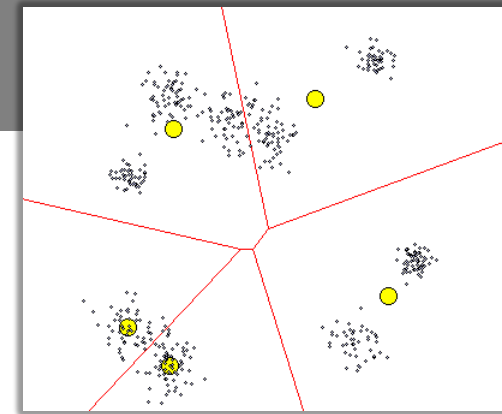
■ Back to K-means

$$\text{Minimize: } J(D, W) := \sum_{w_j} \sum_{\{x_i | w(x_i)=w_j\}} \|x_i - w_j\|^2$$

$$\text{where } w(x_i) := w_k \mid k = \operatorname{argmin}_j \{\|x_i - w_j\|\}$$

$$\frac{\partial J(D, W)}{\partial w_j} = 2 \sum_{\{x_i | w(x_i)=w_j\}} (w_j - x_i)$$

Keeping the assignment $w(x)$ constant

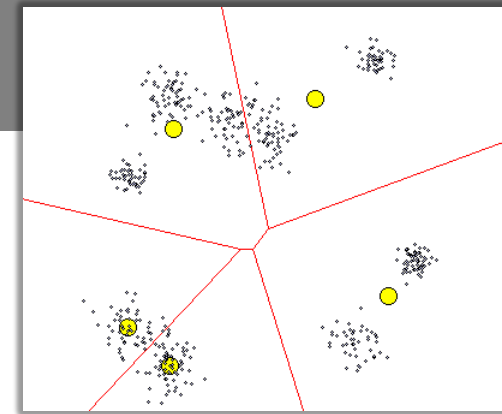


Stochastic Gradient Descent

■ Back to K-means

$$\text{Minimize: } J(D, W) := \sum_{w_j} \sum_{\{x_i | w(x_i)=w_j\}} \|x_i - w_j\|^2$$

$$\text{where } w(x_i) := w_k \mid k = \operatorname{argmin}_j \{\|x_i - w_j\|\}$$



$$\frac{\partial J(D, W)}{\partial w_j} = 2 \sum_{\{x_i | w(x_i)=w_j\}} (w_j - x_i)$$

Keeping the assignment $w(x)$ constant

■ Alternate optimization

At each iteration, place each *landmark* w_j at the centroid of its assigned samples

$$w_i = \frac{1}{|\{x_i | w(x_i) = w_j\}|} \sum_{\{x_i | w(x_i)=w_j\}} x_i \quad \text{It is obtained by imposing } \frac{\partial J(D, W)}{\partial w_j} = 0$$

In the case of K-means, the above is equivalent to Newton's optimization method [Bottou & Bengio, 1998], which would require:

$$\Delta w_i = - \left[\frac{\partial^2 J(D, W)}{\partial w_i^2} \right]^{-1} \frac{\partial J(D, W)}{\partial w_i}$$

Stochastic Gradient Descent

- *Back to K-means*

Minimize: $J(D, W) := \sum_{w_j} \sum_{\{x_i | w(x_i)=w_j\}} \|x_i - w_j\|^2$

where $w(x_i) := w_k \mid k = \operatorname{argmin}_j \{\|x_i - w_j\|\}$

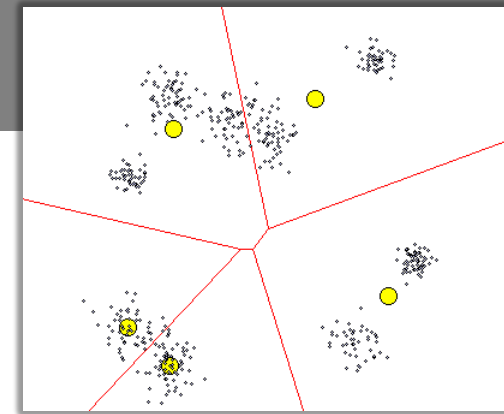
$$\frac{\partial J(D, W)}{\partial w_j} = 2 \sum_{\{x_i | w(x_i)=w_j\}} (w_j - x_i)$$

Keeping the assignment $w(x)$ constant

- *(Batch) Gradient Descent (GD)*

At each iteration, update each *landmark* w_j as

$$\Delta w_i = -\alpha \frac{\partial J(D, W)}{\partial w_j} = \alpha \sum_{\{x_i | w(x_i)=w_j\}} (x_i - w_j)$$

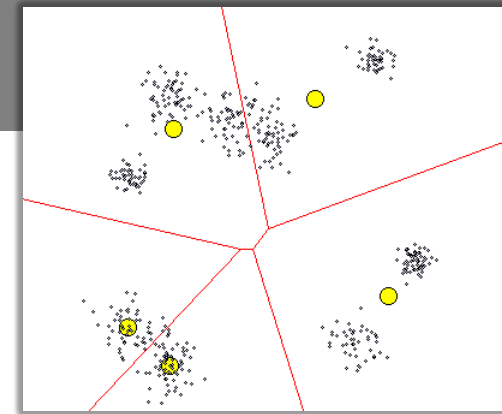


Stochastic Gradient Descent

■ Back to K-means

$$\text{Minimize: } J(D, W) := \sum_{w_j} \sum_{\{x_i | w(x_i)=w_j\}} \|x_i - w_j\|^2$$

$$\text{where } w(x_i) := w_k \mid k = \operatorname{argmin}_j \{\|x_i - w_j\|\}$$



$$\frac{\partial J(D, W)}{\partial w_j} = 2 \sum_{\{x_i | w(x_i)=w_j\}} (w_j - x_i)$$

Keeping the assignment $w(x)$ constant

■ Stochastic Gradient Descent (SGD)

At each iteration, update each landmark w_j as

$$\Delta w_i = -\alpha \frac{\partial J(x_i, W)}{\partial w_j} = \alpha (x_i - w_j), \quad j := \operatorname{argmin}_k \|x_i - w_k\|$$

By taking the average of Δw_i over W :

$$\langle \Delta w_i \rangle = \langle \alpha (x_i - w_j) \rangle = \alpha \frac{1}{|W|} \sum_{\{x_i | w(x_i)=w_j\}} (x_i - w_j)$$

$$= -\alpha' \frac{\partial J(D, W)}{\partial w_j}$$

i.e. on average, the SGD step looks like the Batch GD...

Stochastic Gradient Descent

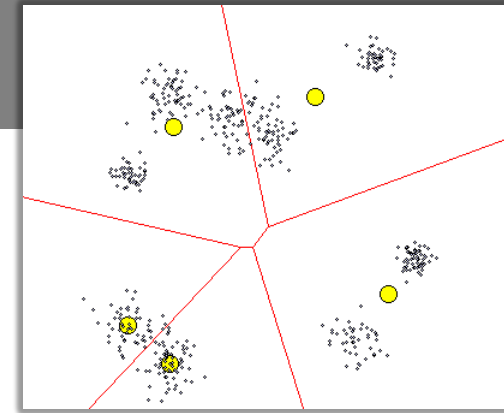
■ *Apropos convergence*

Alternate optimization (and Newton's)

Under safety conditions

$$\hat{W}_n \rightarrow W^* \quad \text{when } n \rightarrow \infty$$

where W^* is a local minimum of $J(D, W)$



(Batch) Gradient Descent (GD)

Under safety conditions and if α is progressively reduced over time

$$\hat{W}_n \rightarrow W^* \quad \text{when } n \rightarrow \infty$$

Stochastic Gradient Descent (SGD)

Under safety conditions and if α is progressively reduced over time

$$\text{SMA}_N[\hat{W}_n] \rightarrow W^* \quad \text{when } n \rightarrow \infty$$

i.e. it converges in the simple moving average, for a suitable N

Stochastic Gradient Descent

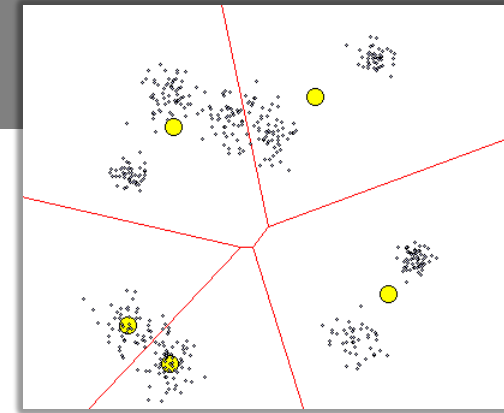
- *Apropos the speed of convergence*

Accuracy ρ is attained when either

$$|\hat{W} - W^*| \leq \rho$$

or

$$|\text{SMA}_N[\hat{W}_n] - W^*| \leq \rho$$

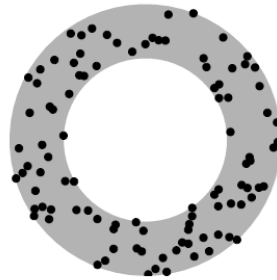


d is the dimension of the data space

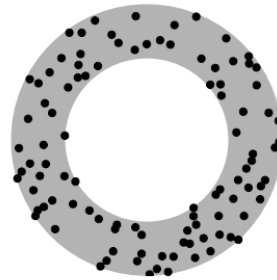
Algorithm	Cost per iteration	Iterations to reach accuracy ρ	Time to reach accuracy ρ
<i>Alternate optimization (AO)</i>	$\mathcal{O}(nd)$	$\mathcal{O}\left(\log \log \frac{1}{\rho}\right)$	$\mathcal{O}\left(nd \log \log \frac{1}{\rho}\right)$
<i>Gradient descent (GD)</i>	$\mathcal{O}(nd)$	$\mathcal{O}\left(\log \frac{1}{\rho}\right)$	$\mathcal{O}\left(nd \log \frac{1}{\rho}\right)$
<i>Stochastic gradient descent (SGD)</i>	$\mathcal{O}(d)$	$\mathcal{O}\left(\frac{1}{\rho}\right)$	$\mathcal{O}\left(d \frac{1}{\rho}\right)$

[from Bottou & Bousquet, 2007]

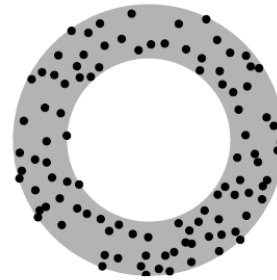
Hard Competitive Learning



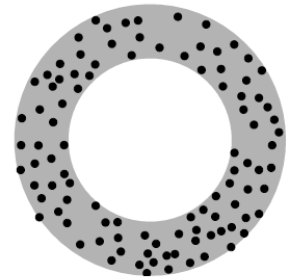
a) 0 signals



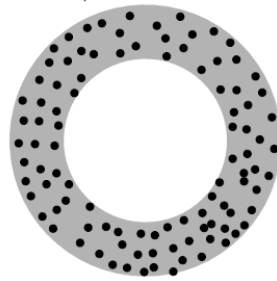
b) 100 signals



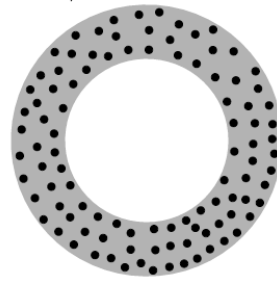
c) 300 signals



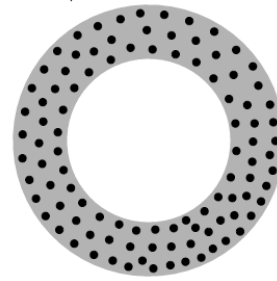
d) 1000 signals



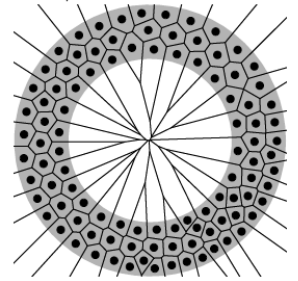
e) 2500 signals



f) 10000 signals



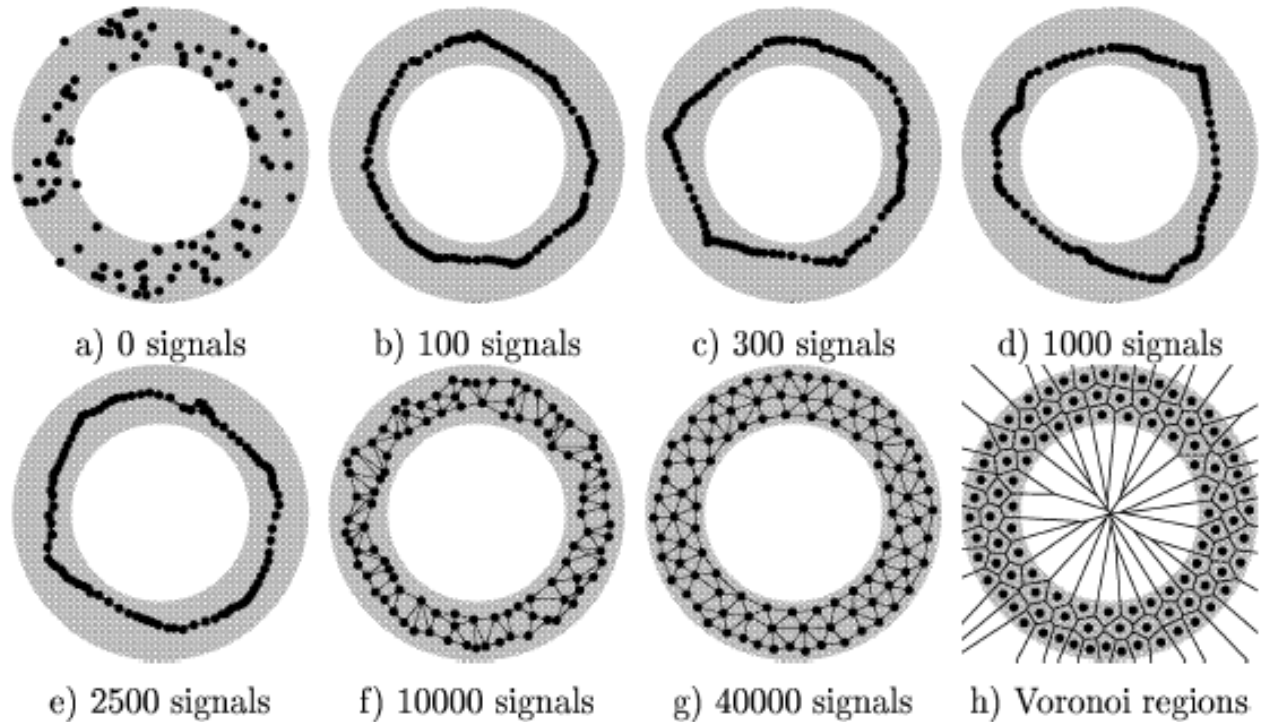
g) 40000 signals



h) Voronoi regions

Neural Gas (Martinetz e Schulten, 1991)

An online algorithm
that is less vulnerable
to local minima



■ Algorithm

A set $\mathbf{W} = \{ u_1, u_2, \dots, u_n \}$ of units, each associated to a vector $w_i \in \mathbf{R}^d$
An input data stream $x \in \mathbf{R}^d$, with probability distribution $P(x)$

- 1) Initialize all vectors w_i at random
- 2) Receive a signal x with probability $P(x)$
- 3) Adapt all vectors w_i to the input signal

$$\Delta w_i = \varepsilon \cdot h_\lambda(k_i(x)) (x - w_i)$$

where ε is the *learning rate*, $k_i(x)$ is a function that assign to unit u_i its *ranking* in \mathbf{W} in terms of distance to v (i.e. the closest unit has ranking 0) and

$$h_\lambda(k_i(x)) := e^{-\frac{k_i(x)}{\lambda}}$$

- 4) Unless some termination criterion is met, go back to step 2)

Note: for $\lambda=0$ the Neural Gas algorithm reduces to online K-Means

- Neural Gas as stochastic learning

Stochastic gradient descent over the function

$$E_{NG}(\mathbf{W}) = \frac{1}{2C_\lambda} \sum_{i=1}^k \int_V P(v) h_\lambda(k_i(v)) (v - w_i)^2 dv$$

$$C_\lambda := \sum_{i=0}^{k-1} h_\lambda(i)$$

- Mean adaptation of units in Neural Gas

$$\langle \Delta w_i \rangle \propto \frac{1}{\rho(w_i)^{1+\frac{2}{d}}} \left(\overset{\text{Input data distribution}}{\partial_r P(w_i)} - \alpha^{-1} P(w_i) \frac{\partial_r \rho(w_i)}{\rho(w_i)} \right)$$

Spatial density of units

Each unit is subject to two 'forces':

- 1) an attractive force towards the regions where signal density is higher
- 2) a repulsive force among units themselves

With uniform input probability, units become uniformly distributed as well (maximum entropy)

Self-Organization

(from Wikipedia)

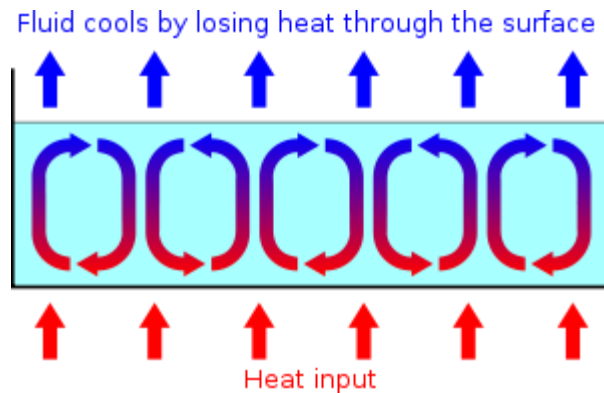
“Self-organization is a process in which the internal organization of a system, normally an open system, increases in complexity without being guided or managed by an outside source.”

Example: Bénard cells (*Self-Organizing System*)

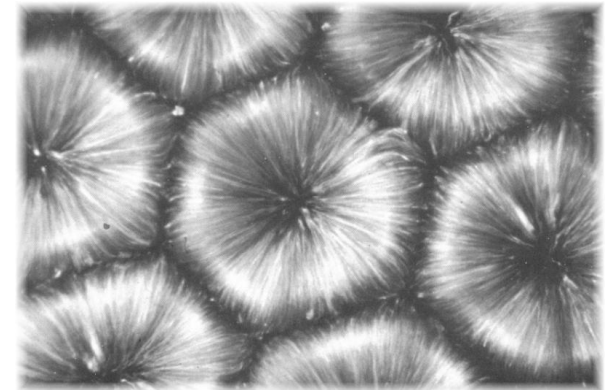
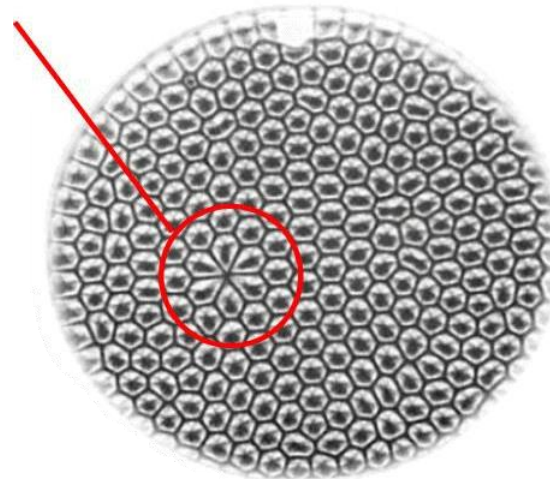
A thin layer of a fluid, heated from below.

As long as the temperature gradient within the layer is small, the fluid remains still and the heat propagates by conduction only.

As the gradient increases, convection flows begin to appear and the flow organizes into cells



An almost perfect pattern



Example: liquid crystals (*Self-Organizing System*)

Molecules having specific shapes exhibit the tendency to organize spontaneously

The phenomenon may depend on temperature (i.e. thermotropic) or on concentration (i.e. lyotropic) or both

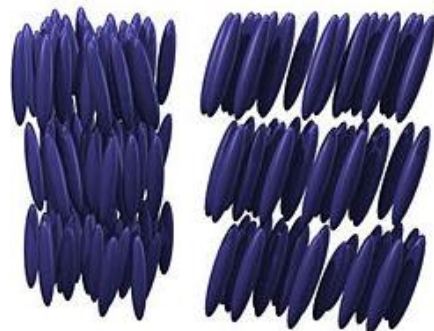
Nematic phase

no positional order
prevalent orientation



Smectic phase

formation of layers
prevalent orientation



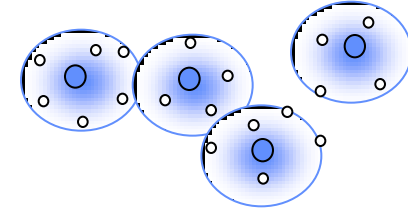
Chiral or cholesteric phase

formation of layers
prevalent orientation
larger scale organization

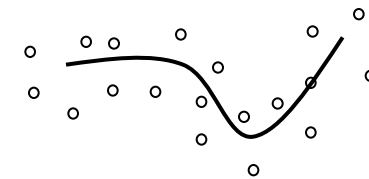
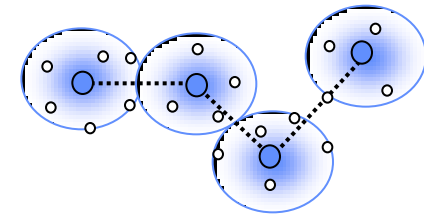
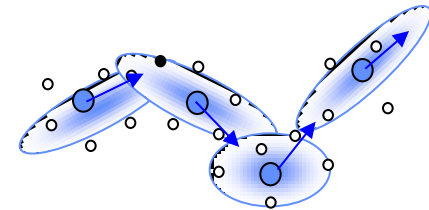


Adapting units or networks? (competing vs cooperation)

- Unit (vector) adaptation (i.e. for *clustering*)



- Network structure



Self-Organizing Maps (SOM) (Kohonen, 1985)

■ A network with fixed topology

A set $\mathbf{W} = \{ u_1, u_2, \dots, u_n \}$ of units, each associated to a vector $w_i \in \mathbf{R}^d$

A set of connections $\mathbf{C} : \mathbf{W} \times \mathbf{W}$ with a predefined topology (typically a *grid*)

An input data stream $x \in \mathbf{R}^d$, with probability distribution $P(\xi)$

1) Initialize all vectors w_i at random

2) Receive a signal x with probability $P(x)$

3) Determine the *winning unit*

$$b = \operatorname{argmin}_k \|x - w_k\|$$

4) Adapt all vectors w_i to the input signal

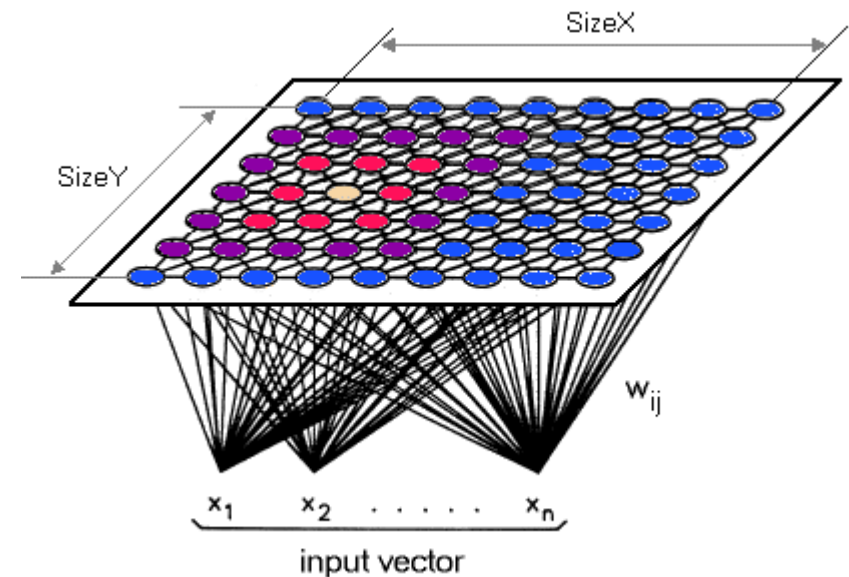
$$\Delta w_i = \varepsilon(t) h_\lambda(i, b) (x - w_i)$$

where $\varepsilon(t)$ is a time-varying *learning rate* and

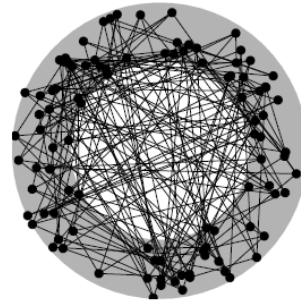
$h(i, b)$ is defined as

$$h(i, b) := e^{\frac{-d(i, b)}{\lambda(t)}} \quad \text{--- block distance on the grid } \mathbf{C}$$

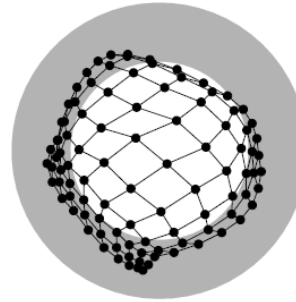
5) Unless some termination criterion is met, go back to step 2)



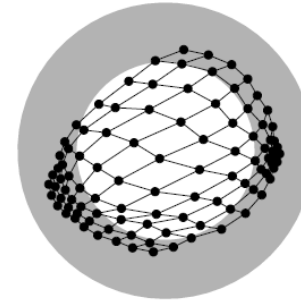
Self-Organizing Maps (SOM)



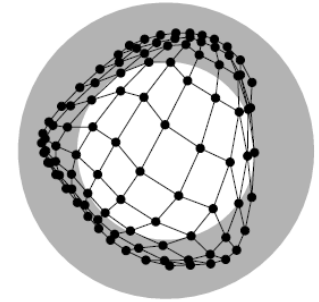
a) 0 signals



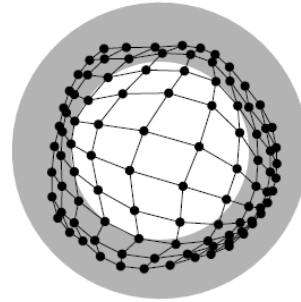
b) 100 signals



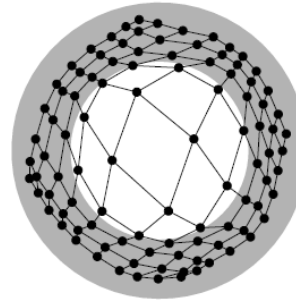
c) 300 signals



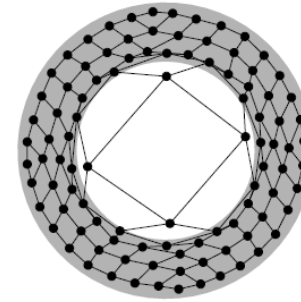
d) 1000 signals



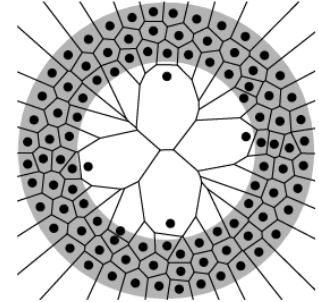
e) 2500 signals



f) 10000 signals



g) 40000 signals



h) Voronoi regions

A SOM performs no 'classical' optimization

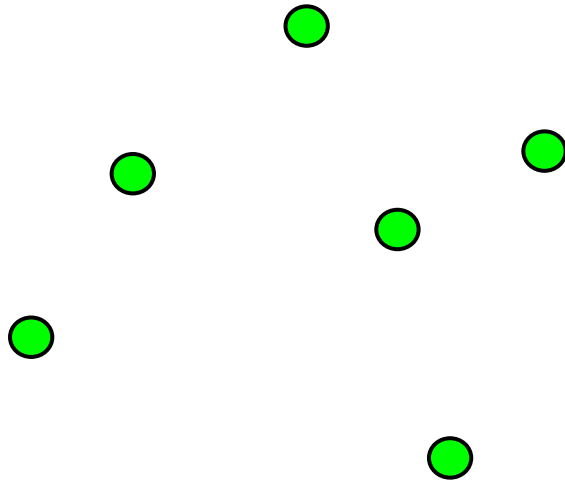
There is no objective function that a SOM optimizes

[E. Erwin, K. Obermayer, and K. Schulten, 1992]

Voronoi and Delaunay

- **Voronoi tessellation**

Starting from a set of *landmarks*



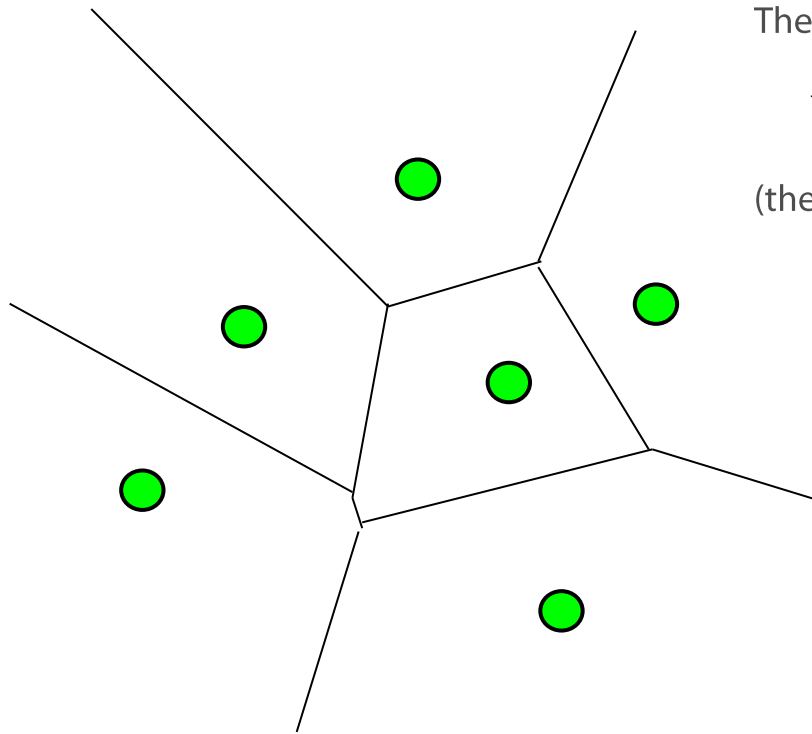
Voronoi and Delaunay

■ Voronoi tessellation

Starting from a set of *landmarks*

Define the **Voronoi regions** associated to each *landmark*

The **Voronoi tessellation** (in \mathbf{R}^d) is the union of all Voronoi regions



The Voronoi region associated to a landmark w_i

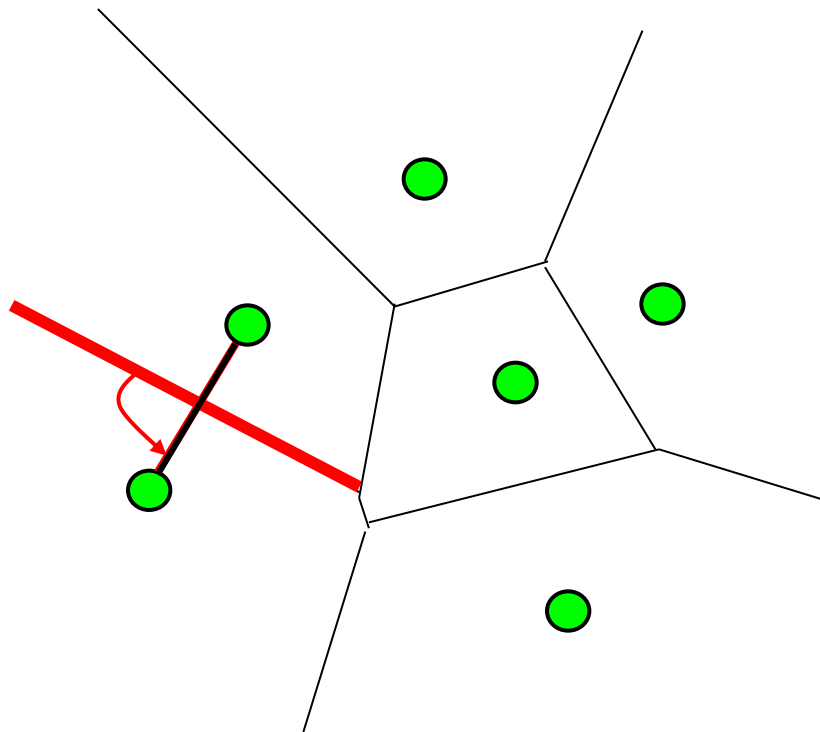
$$V_i := \left\{ x \in \mathbf{R}^d \mid \|x - w_i\| \leq \|x - w_j\|, \forall j \neq i \right\}$$

(the boundaries are *shared* among regions)

Voronoi and Delaunay

▪ Delaunay graph

Define an arc between each two landmarks whose region have non-empty intersections

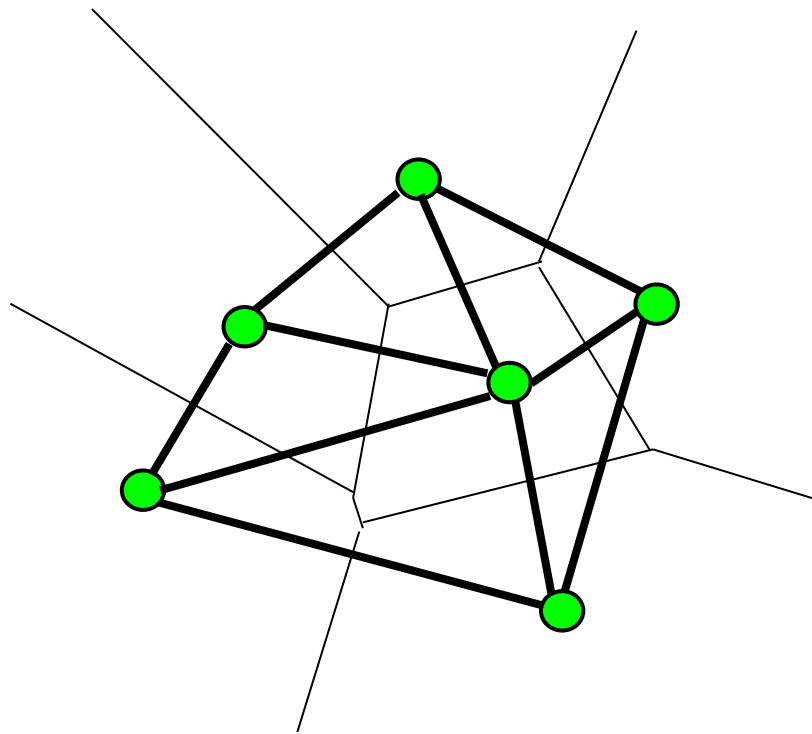


Each arc in the graph is orthogonal to the boundary between the two corresponding regions

Voronoi and Delaunay

▪ Delaunay graph

Define an arc between each two landmarks whose region have non-empty intersections
The **Delaunay graph** is the union of all landmarks (*nodes*) and the arcs joining landmarks whose Voronoi regions have non-empty intersections



The Delaunay graph is a triangulation that maximizes the minimal angle of triangles

Hebbian learning and Delaunay graph

■ Algorithm

A set $\mathbf{W} = \{ u_1, u_2, \dots, u_n \}$ of units, each associated to a vector $w_i \in \mathbf{R}^d$ **at fixed positions**

An input data stream $x \in \mathbf{R}^d$, with probability distribution $P(x)$

- 1) Receive a signal x with probability $P(x)$
- 2) Find the indexes b and s of the closest and second-closest units to x and add (b, s) to \mathbf{C}
- 3) Unless some termination criterion is met, go back to step 1)

At all times, this algorithm produces a graph which is a subset of the Delaunay graph over the units in \mathbf{W} . In the limit, under some conditions on $P(x)$, it produces the Delaunay graph

[de Silva & Carlsson, 2004]

Neural Gas with Hebbian Learning

■ Algorithm

Differences with the NG algorithm are in red

A set $\mathbf{W} = \{ u_1, u_2, \dots, u_n \}$ of units, each associated to a vector $w_i \in \mathbf{R}^d$

A set of connections \mathbf{C} , initially empty; each connection has an *age* value

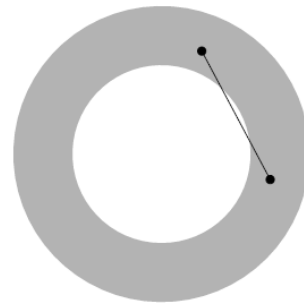
An input data stream $x \in \mathbf{R}^d$, with probability distribution $P(x)$

- 1) Initialize all vectors \mathbf{p}_i at random
- 2) Receive a signal x with probability $P(x)$
- 3) Find the indexes b and s of the closest and second-closest units to x and add (b, s) to \mathbf{C} , setting its *age* to 0
- 4) Adapt all vectors w_i to the input signal (see Neural Gas)

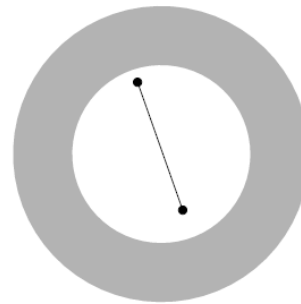
$$\Delta w_i = \varepsilon \cdot h_\lambda(k_i(x)) (x - w_i)$$

- 5) Increase the *age* of all connections in \mathbf{C} by 1 and remove those beyond a certain threshold T_{age} . Remove units that remain isolated
- 6) Unless some termination criterion is met, go back to step 2)

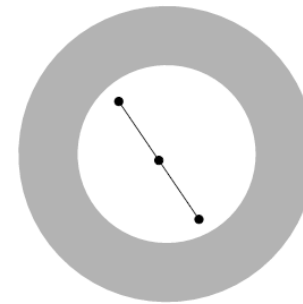
Growing Neural Gas [Fritzke, 1995]



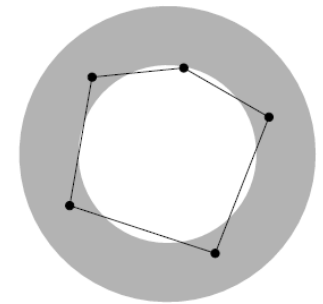
a) 0 signals



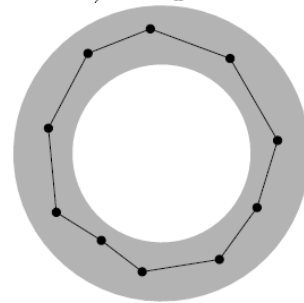
b) 100 signals



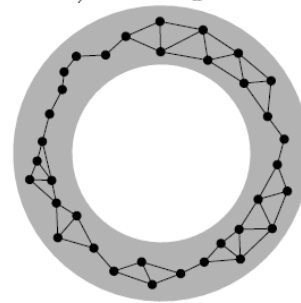
c) 300 signals



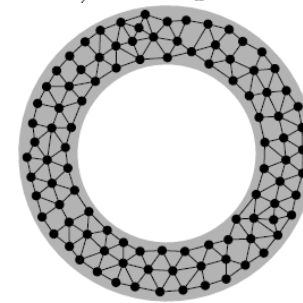
d) 1000 signals



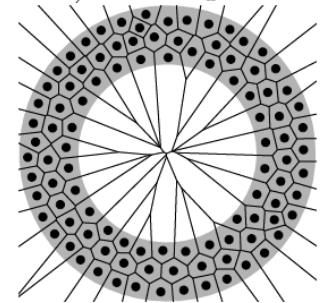
e) 2500 signals



f) 10000 signals



g) 40000 signals



h) Voronoi regions

Growing Neural Gas (GNG)

■ Algorithm

Differences with the NG + Hebbian Learning algorithm are in red

A set $\mathbf{W} = \{ u_1, u_2, \dots, u_n \}$ of units, each associated to a vector $w_i \in \mathbf{R}^d$

A set of connections \mathbf{C} , initially empty; each connection has an *age* value

An input data stream $x \in \mathbf{R}^d$, with probability distribution $P(x)$

- 1) Initialize all vectors w_i at random
- 2) Receive a signal x with probability $P(x)$
- 3) Find the indexes b and s of the closest and second-closest units to x and add (b, s) to \mathbf{C} , setting its *age* to 0
- 4) Adapt the vectors **of the closest unit and its immediate neighbors** by

$$\Delta w_b = \varepsilon_b \cdot (x - w_b)$$

$$\Delta w_i = \varepsilon_i \cdot (x - w_i) \quad \forall i \mid (b, i) \in \mathbf{C} \quad \varepsilon_i \ll \varepsilon_b$$

- 5) **Every λ iterations, find the unit with the greatest accumulated error and its neighbor with the second-greatest error and create a *new unit* in between**
- 6) Increase the *age* of all connections in \mathbf{C} by 1 and remove those beyond a certain threshold T_{age} . Remove units that remain isolated
- 7) Unless some termination criterion is met, go back to step 2)

Grow-when-required networks [Marsland, 2002]

■ Algorithm

Differences with the GNG algorithm are in red

A set $\mathbf{W} = \{ u_1, u_2, \dots, u_n \}$ of units, each associated to a vector $w_i \in \mathbf{R}^d$

A set of connections \mathbf{C} , initially empty; each connection has an *age* value

An input data stream $x \in \mathbf{R}^d$, with probability distribution $P(x)$

1) Initialize all vectors w_i at random

2) Receive a signal x with probability $P(x)$

3) Find the indexes b and s of the closest and second-closest units to x and add (b, s) to \mathbf{C} , setting its *age* to 0

4) If x is farther away from w_b than a given radius r create a new unit and connect it with both b and s

Otherwise, adapt the vectors of the closest unit and its immediate neighbors by

$$\Delta w_b = \varepsilon_b \cdot (x - w_b)$$

$$\Delta w_i = \varepsilon_i \cdot (x - w_i) \quad \forall i | (b, i) \in \mathbf{C} \quad \varepsilon_i \ll \varepsilon_b$$

5) Increase the *age* of all connections in \mathbf{C} by 1 and remove those beyond a certain threshold T_{age} . Remove units that remain isolated

6) Unless some termination criterion is met, go back to step 2)

Learning topologies

GNG and GWR networks can adapt their topology to the input data

