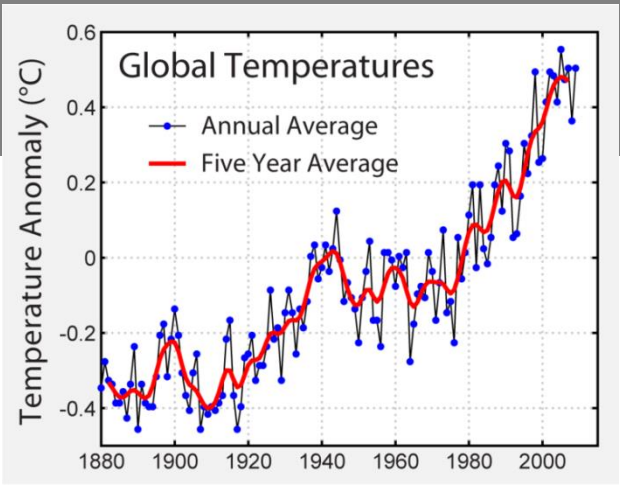
Self-organizing systems

Marco Piastra

Moving averages



[image from wikipedia]

(Just an aside about following non-stationary phenomena)

- Average

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

  Running implementation:

$$\overline{v}_T = \frac{1}{T}(v_T + \sum_{k=1}^{T-1} v_k) = \frac{1}{T}(v_T + (T-1)\overline{v}_{T-1})$$

$$= \overline{v}_{T-1} + \frac{1}{T}(v_T - \overline{v}_{T-1}) = \frac{1}{T} v_T + (1 - \frac{1}{T})\,\overline{v}_{T-1}$$

"the weight of newer observations diminishes with time"

- Simple Moving Average (SMA)

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

- Exponential Moving Average (EMA)

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

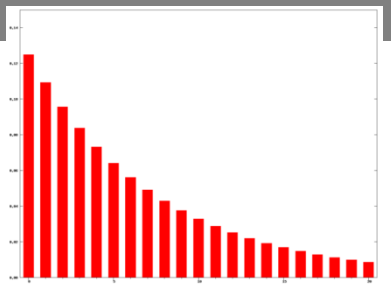
"the weight of newer observations remains constant"

Moving averages

- Exponential Moving Average (EMA)

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

"the weight of older observations diminishes with time"



[image from wikipedia]

Expanding:

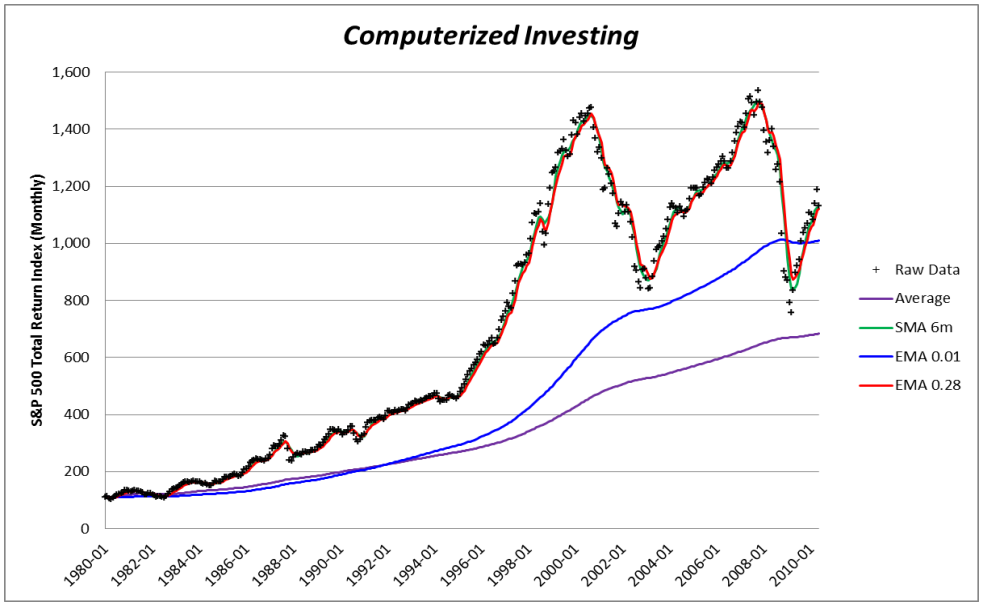
$$\begin{aligned}
\overline{v}_{t,\alpha} &= \alpha\, v_t + (1-\alpha)\, \overline{v}_{t-1,\alpha} \\
&= \alpha\, v_t + (1-\alpha)(\alpha\, v_{t-1} + (1-\alpha)\overline{v}_{t-2,\alpha}) \\
&= \alpha\, v_t + (1-\alpha)(\alpha\, v_{t-1} + (1-\alpha)(\alpha\, v_{t-2} + (1-\alpha)\overline{v}_{t-3,\alpha}))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

The weight of past contributions decays as

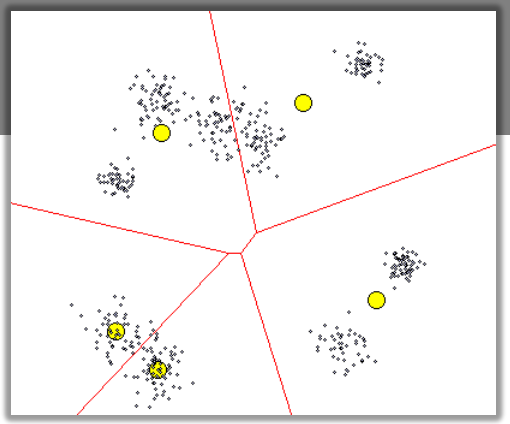
$$(1-\alpha)^{\Delta_t}\, v_{t-\Delta_t}$$

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

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



Stochastic Gradient Descent



- *Back to K-means*

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

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

  Keeping $w(x)$ constant

- Alternate optimization

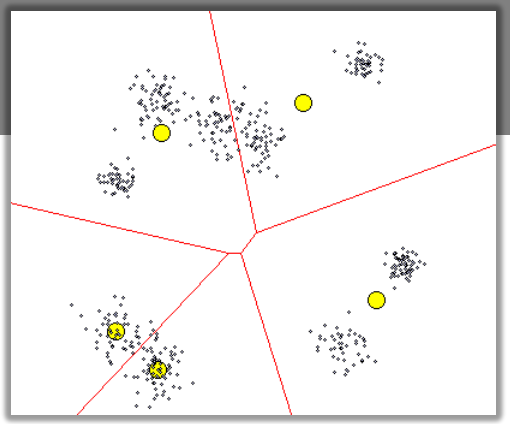
  At each iteration, update each *landmark* w_j by

  $$w_i = \frac{1}{|\{x_i | w(x_i) = w_j\}|} \sum_{\{x_i | w(x_i)=w_j\}} x_i$$

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

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

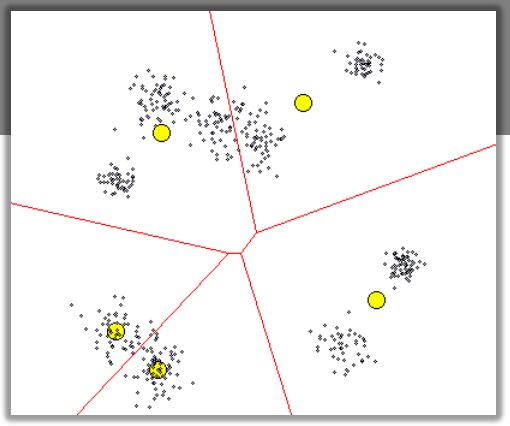
  Keeping $w(x)$ constant

- (*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*

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

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

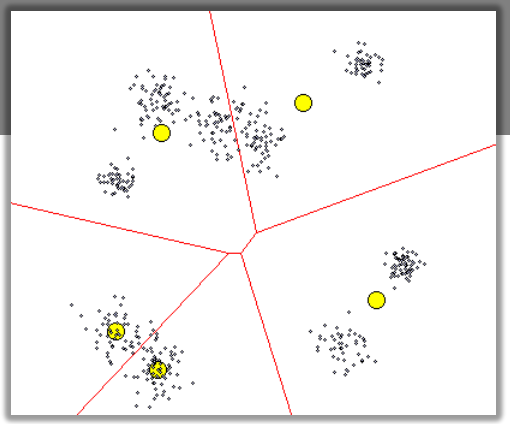
  Keeping $w(x)$ constant

- *Stochastic* Gradient Descent (SGD)

  At each iteration, update ~~each~~ *landmark* w_j as

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

Stochastic Gradient Descent



- *Apropos convergence*

  Alternate optimization (i.e. Newton's)

  Under safety conditions

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

  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*

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

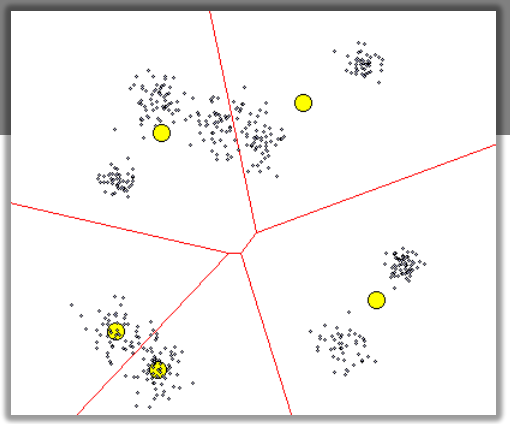
  Stochastic Gradient Descent (SGD)

  Under safety conditions and if α *is progressively reduced over time*

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

  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

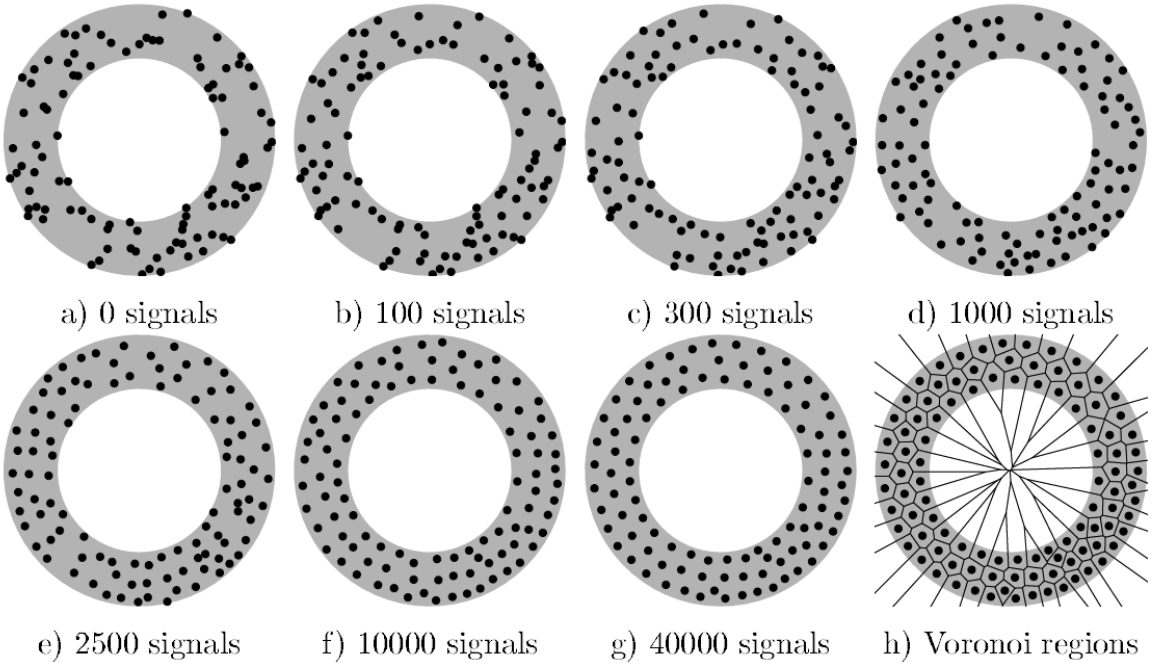
  $$|\mathrm{SMA}_N[\hat{W}_n] - W^*| \leq \rho$$

d *is the dimension of the data space*

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

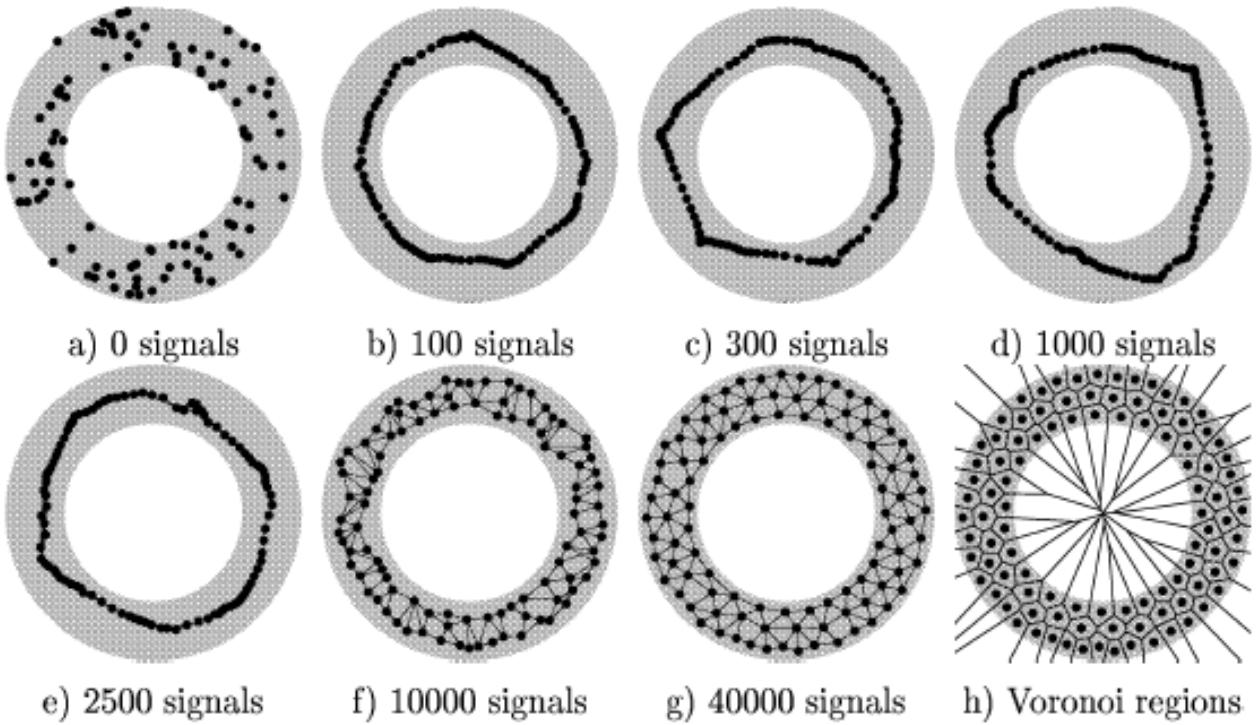
[*from Bottou & Bousquet, 2007*]

Hard Competitive Learning



Neural Gas (Martinetz e Schulten, 1991)

An online algorithm that is less vulnerable to local minima



Neural Gas

- Algorithm

A set $\mathbf{A} = \{ u_1, u_2, \ldots, u_n \}$ of units, each associated to a vector $\mathbf{p}_i \in \mathbf{R}^d$
An input data stream $\boldsymbol{\xi} \in \mathbf{R}^d$, with probability distribution $P(\boldsymbol{\xi})$

1) Initialize all vectors $\mathbf{p}_i$ at random
2) Receive a signal $\boldsymbol{\xi}$ with probability $P(\boldsymbol{\xi})$
3) Adapt *all* vectors $\mathbf{p}_i$ to the input signal

$$\Delta \boldsymbol{p}_i = \varepsilon \cdot h(k(u_i)) \, (\xi - \boldsymbol{p}_i)$$

where ε is the *learning rate*, $k_i(\boldsymbol{\xi})$ is a function that assign to unit u_i its *ranking* in $\mathbf{A}$ in terms of distance to $\boldsymbol{\xi}$ (i.e. the closest unit has ranking 0) and $h(k_i(\boldsymbol{\xi}))$ is defined as:

$$h(k_i(\xi)) := e^{\frac{-k_i(\xi)}{\lambda}}$$

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

- Neural Gas as stochastic learning

  Stochastic gradient descent over the function

$$E_{NG}(A) = \frac{1}{2H} \sum_{i=1}^{n} \int_M h(k_i(\xi))(\xi - \mathbf{p}_i)^2 P(\xi) d\xi$$

$$H = \sum_{i=1}^{n} h(k_i(\xi)).$$

- Mean adaptation of units in Neural Gas

$$\langle \Delta \mathbf{p}_i \rangle \propto \frac{1}{\rho^{5/3}} \nabla P(\xi) - \frac{5}{3} \frac{P}{\rho} \nabla \rho(\xi)$$

Input data distribution

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)

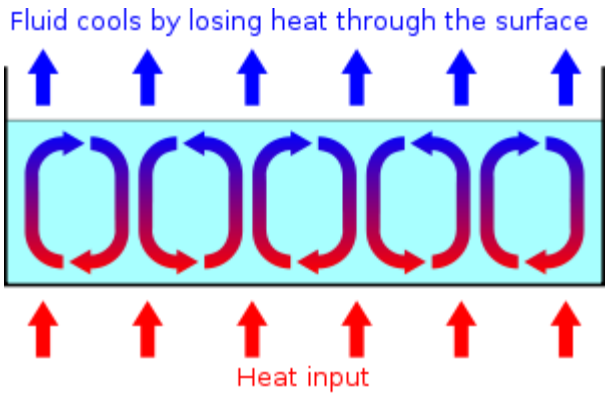
"Self-organization is a process in which the internal organization of a system, normally an open system, increases in complexity without being guided or managed by an outside source."

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

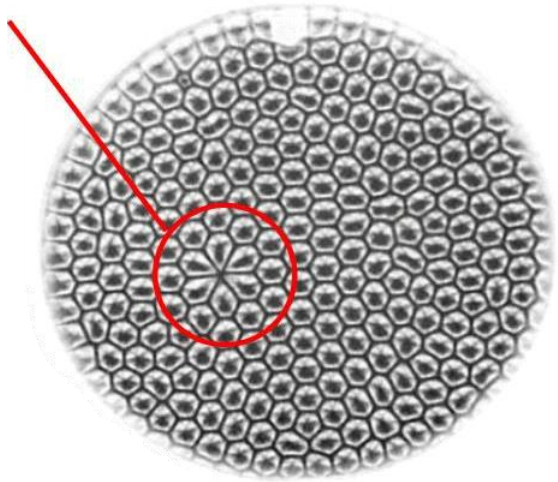
A thin layer of a fluid, heated from below.

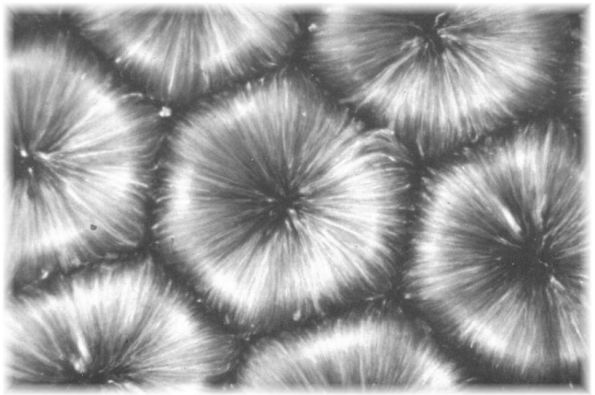
As long as the temperature gradient within the layer is small, the fluid remains still and the heat propagates by conduction only.

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



An almost perfect pattern





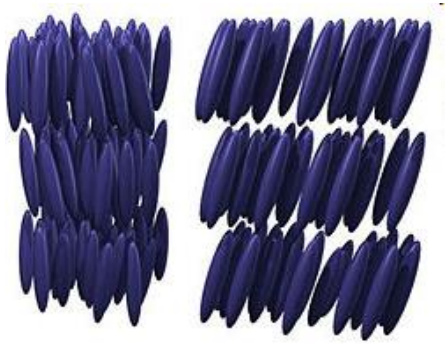
Example: liquid crystals (*Self-Organizing System*)

Molecules having specific shapes exhibit the tendency to organize spontaneously

The phenomenon may depend on temperature (i.e. termotropic) or on concentration (i.e. liotropic) or both

Nematic phase
no positional order
prevalent orientation

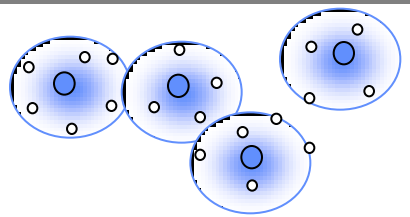
Smectic phase
formation of layers
prevalent orientation

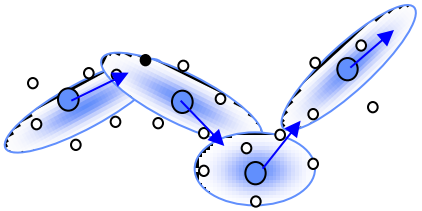


Chiral or cholesteric phase
formation of layers
prevalent orientation
larger scale organization

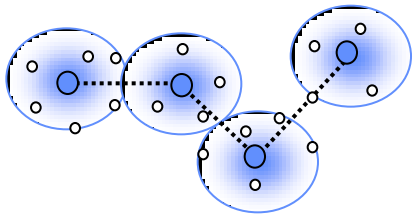
Adapting units or networks? (competing vs cooperation)

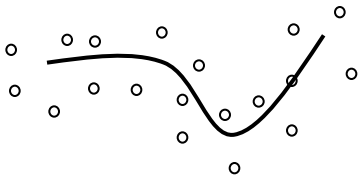
- Unit (vector) adaptation (i.e. for *clustering*)





- Network structure





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

- A network with fixed topology

  A set $\mathbf{A} = \{ u_1, u_2, \ldots, u_n \}$ of units, each associated to a vector $\mathbf{p}_i \in \mathbf{R}^d$
  A set of connections $\mathbf{C} : \mathbf{A} \times \mathbf{A}$ with a predefined topology (typically a *grid*)
  An input data stream $\boldsymbol{\xi} \in \mathbf{R}^d$, with probability distribution $P(\boldsymbol{\xi})$

  1) Initialize all vectors $\mathbf{p}_i$ at random
  2) Receive a signal $\boldsymbol{\xi}$ with probability $P(\boldsymbol{\xi})$
  3) Determine the *winning unit*

$$b = \arg\min_k \| \xi - \boldsymbol{p}_k \|$$

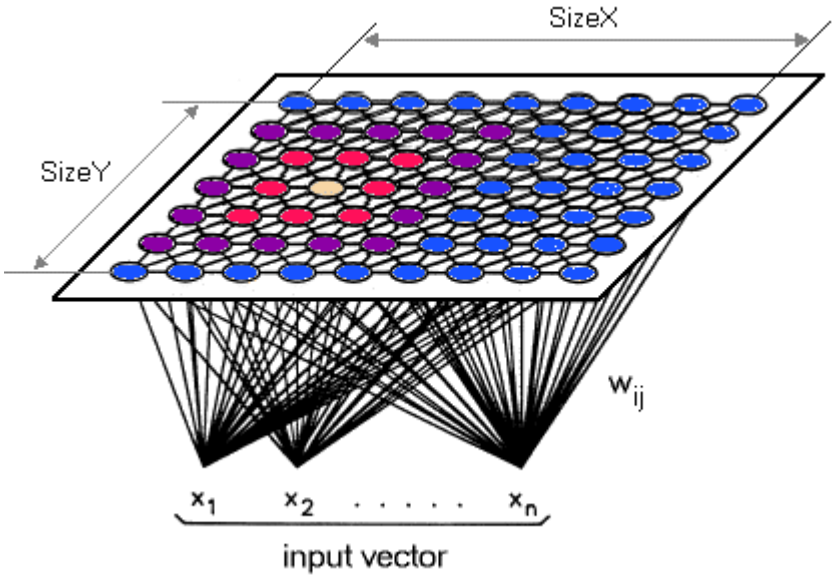
  4) Adapt *all* vectors $\mathbf{p}_i$ to the input signal

$$\Delta \boldsymbol{w}_i = \varepsilon(t)\, h(i,b)\, (\xi - \boldsymbol{w}_i)$$

  where $\varepsilon(t)$ is a time-varying *learning rate* and $h(i,b)$ is defined as

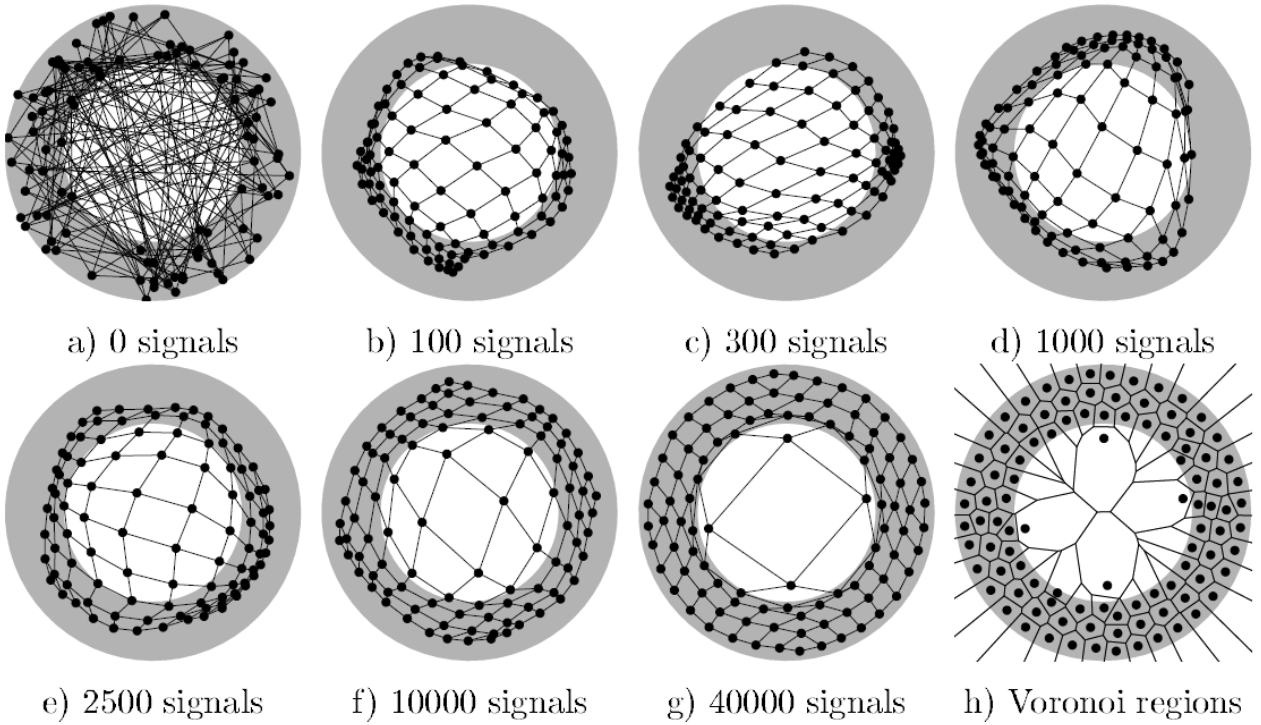
$$h(i,b) := e^{\frac{-d(i,b)}{\lambda(t)}}$$

  block distance on the grid C



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

Self-Organizing Maps (SOM)



A SOM performs no 'classical' optimization

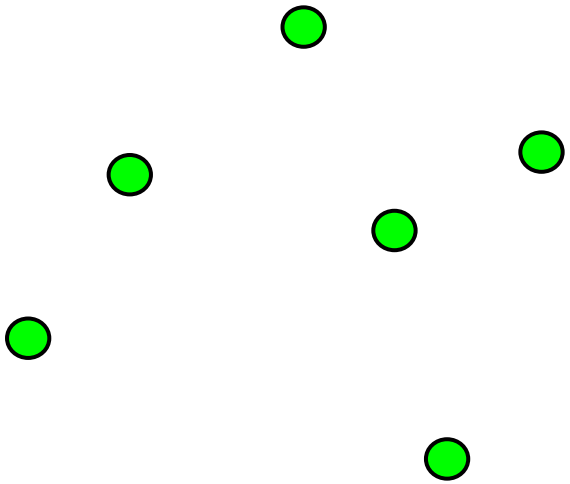
There is no objective function that a SOM optimizes

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

Voronoi and Delaunay

- **Voronoi tessellation**

  Starting from a set of *landmarks*



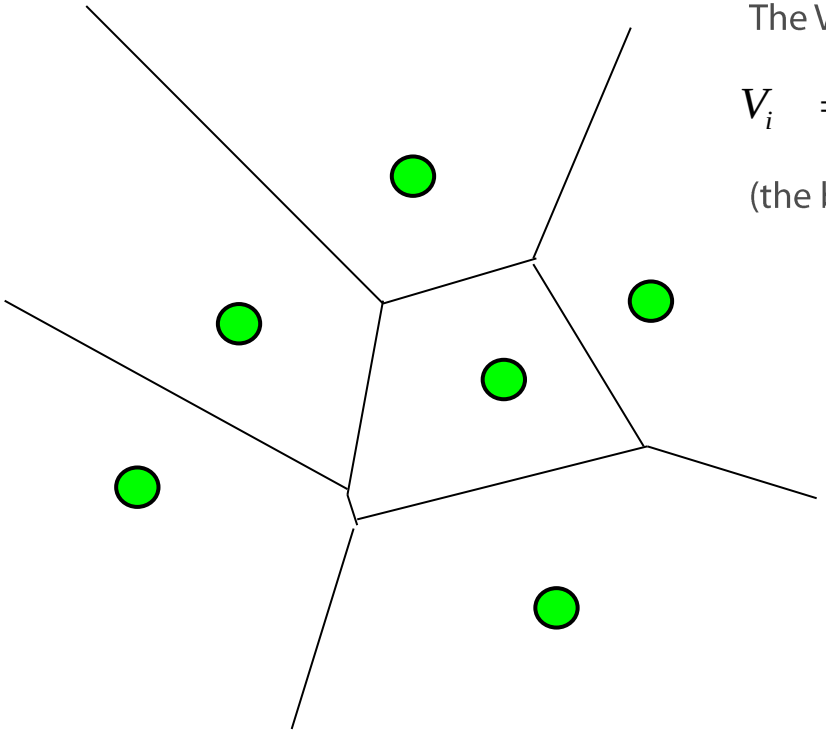
Voronoi and Delaunay

- **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 $\boldsymbol{p}_i$

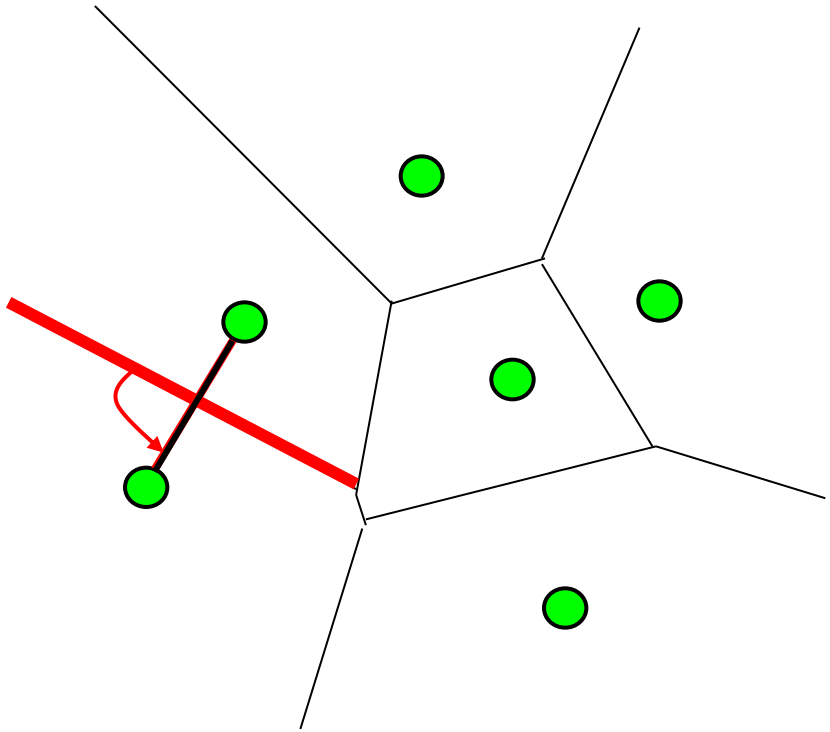
$$V_i = \left\{ \mathbf{x} \in \mathbf{R}^d \;\middle|\; i = \arg\min_j \left\| \mathbf{x} - \mathbf{p}_j \right\| \right\}$$

(the boundaries are *shared* among regions)

Voronoi and Delaunay

- **Delaunay graph**

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



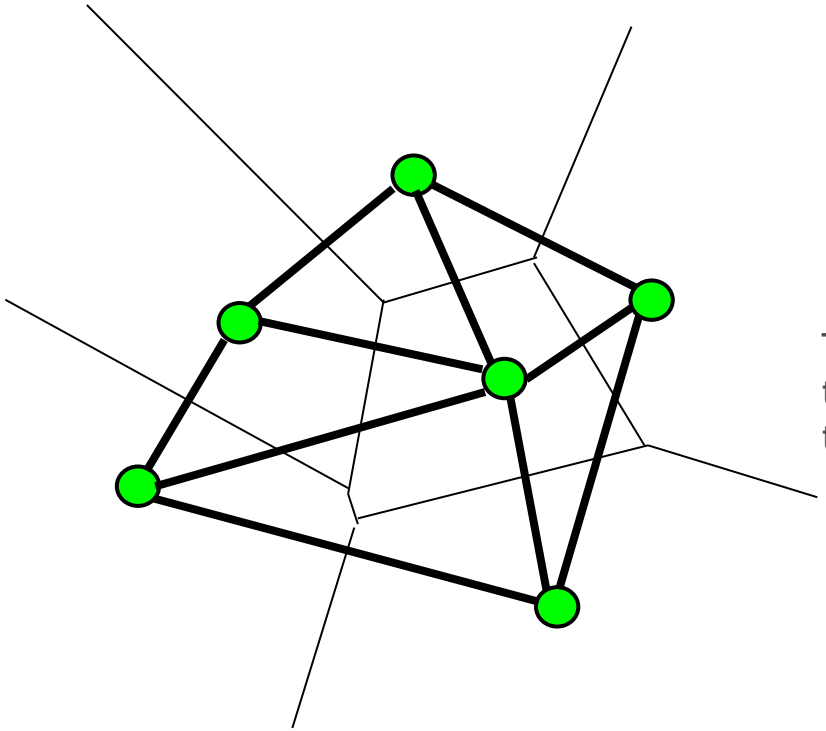
Each arc in the graph is orthogonal to the boundary between the two corresponding regions

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 $\mathbf{A} = \{ u_1, u_2, \ldots, u_n \}$ of units, each associated to a vector $\mathbf{p}_i \in \mathbf{R}^d$ at given positions
  An input data stream $\xi \in \mathbf{R}^d$, with probability distribution $P(\xi)$

  1) Receive a signal ξ with probability $P(\xi)$
  2) Find the indexes b and s of the closest and second-closest units to ξ 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 $\mathbf{A}$. *In the limit, under some conditions on* $P(\xi)$, *it produces the Delaunay graph*
[de Silva & Carlsson, 2004]

Neural Gas with Hebbian Learning

- Algorithm

Differences with the NG algorithm are in red

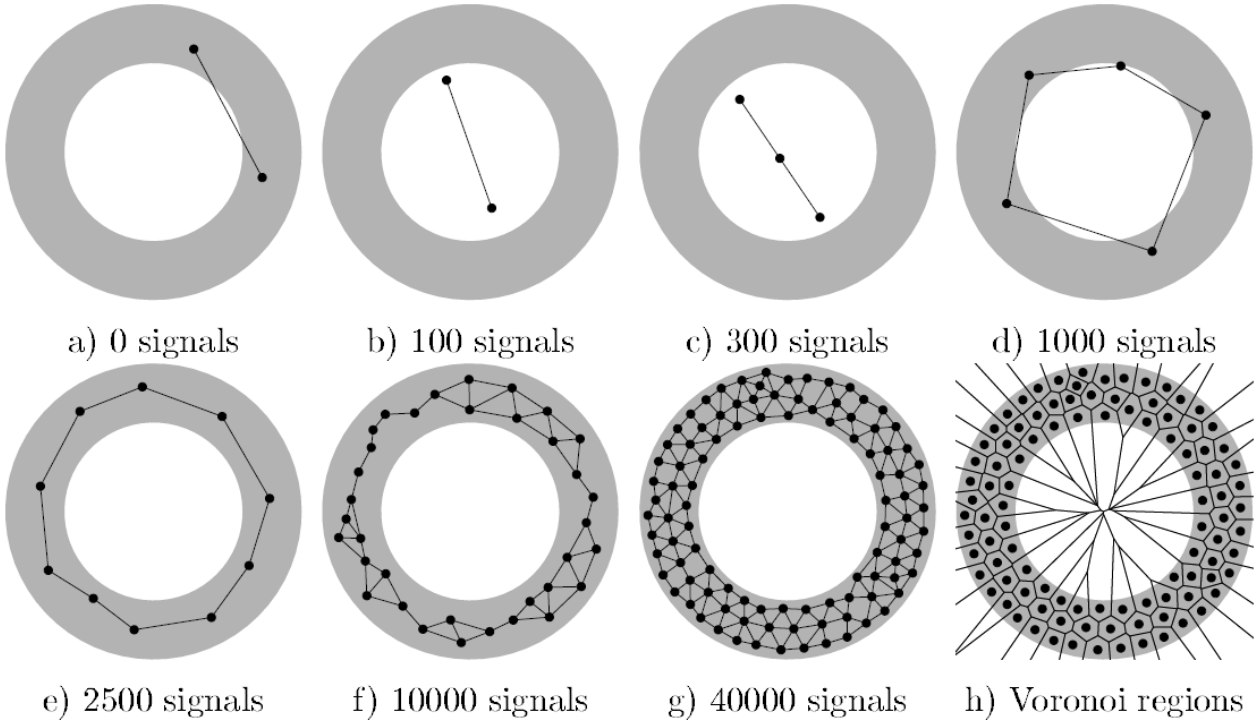
A set $\mathbf{A} = \{ u_1, u_2, \ldots, u_n \}$ of units, each associated to a vector $\mathbf{p}_i \in \mathbf{R}^d$
A set of connections $\mathbf{C}$, initially empty; each connection has an *age* value
An input data stream $\boldsymbol{\xi} \in \mathbf{R}^d$, with probability distribution $P(\boldsymbol{\xi})$

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

$$\Delta \boldsymbol{p}_i = \varepsilon \cdot h(k(u_i)) \, (\xi - \boldsymbol{p}_i)$$

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]



Growing Neural Gas (GNG)

- Algorithm

Differences with the NG + Hebbian Learning algorithm are in red

A set $\mathbf{A} = \{ u_1, u_2, \ldots, u_n \}$ of units, each associated to a vector $\mathbf{p}_i \in \mathbf{R}^d$
A set of connections $\mathbf{C}$, initially empty; each connection has an *age* value
An input data stream $\boldsymbol{\xi} \in \mathbf{R}^d$, with probability distribution $P(\boldsymbol{\xi})$

1) Initialize all vectors $\mathbf{p}_i$ at random
2) Receive a signal $\boldsymbol{\xi}$ with probability $P(\boldsymbol{\xi})$
3) Find the indexes b and s of the closest and second-closest units to $\boldsymbol{\xi}$ 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

$$\Delta \boldsymbol{p}_b = \varepsilon_b \cdot (\xi - \boldsymbol{p}_b)$$
$$\Delta \boldsymbol{p}_i = \varepsilon_i \cdot (\xi - \boldsymbol{p}_i) \quad \forall i \,|\, (b,i) \in \mathrm{C} \quad \varepsilon_i << \varepsilon_b$$

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 $\mathbf{A} = \{ u_1, u_2, \ldots, u_n \}$ of units, each associated to a vector $\mathbf{p}_i \in \mathbf{R}^d$
A set of connections $\mathbf{C}$, initially empty; each connection has an *age* value
An input data stream $\boldsymbol{\xi} \in \mathbf{R}^d$, with probability distribution $P(\boldsymbol{\xi})$

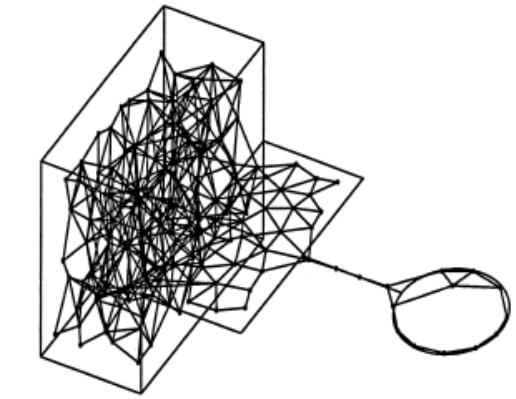
1) Initialize all vectors $\mathbf{p}_i$ at random
2) Receive a signal $\boldsymbol{\xi}$ with probability $P(\boldsymbol{\xi})$
3) Find the indexes b and s of the closest and second-closest units to $\boldsymbol{\xi}$ and add (b, s) to $\mathbf{C}$, setting its *age* to 0
4) If the $\boldsymbol{\xi}$ is farther away from $\mathbf{p}_b$ 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

$$\Delta \boldsymbol{p}_b = \varepsilon_b \cdot (\xi - \boldsymbol{p}_b)$$
$$\Delta \boldsymbol{p}_i = \varepsilon_i \cdot (\xi - \boldsymbol{p}_i) \quad \forall i \,|\, (b,i) \in \mathrm{C} \quad \varepsilon_i << \varepsilon_b$$

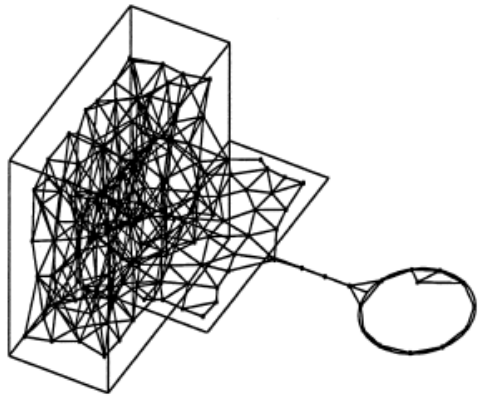
5) Increase the *age* of all connections in $\mathbf{C}$ by 1 and remove those beyond a certain threshold T_{age}. Remove units that remain isolated
6) Unless some termination criterion is met, go back to step 2)

Learning topologies

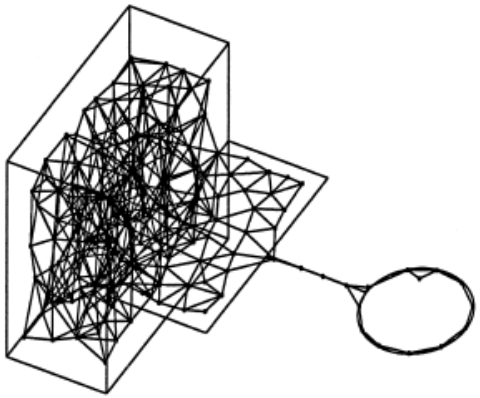
GNG and GWR networks can adapt their topology to the input data



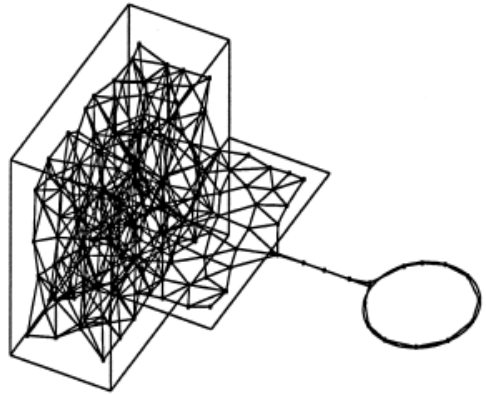
5000 input signals
172 nodes
546 edges



10000 input signals
184 nodes
591 edges



15000 input signals
191 nodes
620 edges



20000 input signals
193 nodes
631 edges