Intelligenza Artificiale II

Self-Organizing Networks

Marco Piastra

Self-Organization

- (*Wikipedia*)

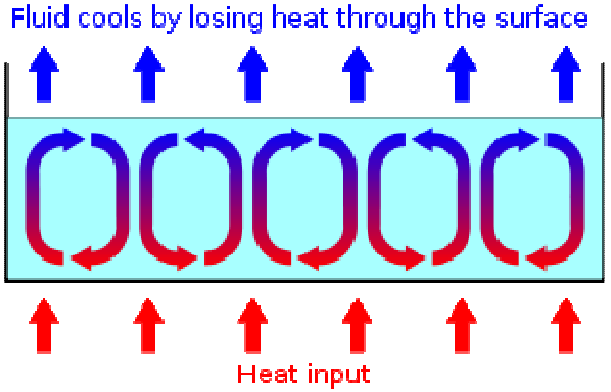
"Self-organization is a process in which the internal organization of a system, normally an open system, increases in complexity without being guided or managed by an outside source."

Esempio: celle di Bénard (*Self-Organizing System*)

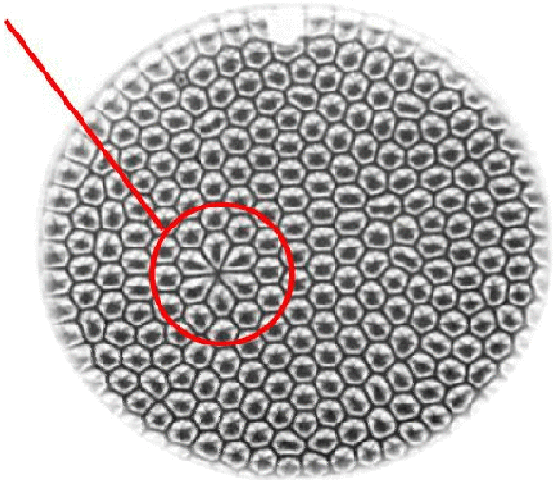
Un fluido in uno strato sottile, riscaldato uniformemente dal basso.

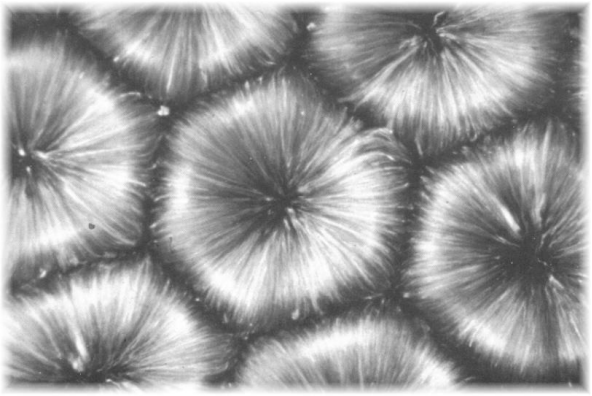
Finché il gradiente di temperatura è molto ridotto,
il fluido rimane immobile ed il calore si propaga per conduzione.

All'aumentare del gradiente, si generano dei flussi di convezione
che si organizzano spontaneamente in *celle*



Un pattern *quasi* perfetto



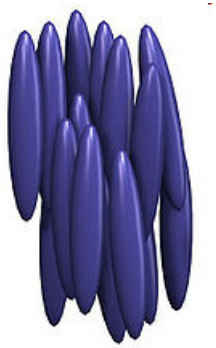


Esempio: cristalli liquidi (*Self-Organizing System*)

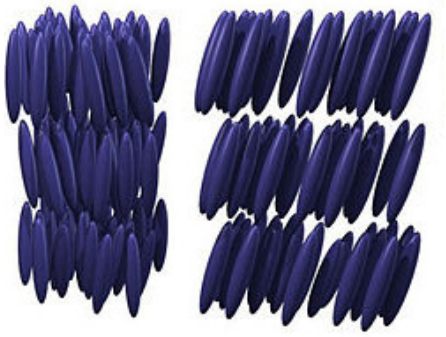
Molecole aventi forme particolari presentano una tendenza ad organizzarsi spontaneamente

Il fenomeno può dipendere dalla temperatura (termotropico) o dalla concentrazione (liotropico) o da entrambe

Fase **nematica**
nessun ordine posizionale
orientamento prevalente



Fasi **smectica**
formazione di strati
orientamento prevalente



Fasi **chirali** o **colesteriche**
formazione di strati
orientamento prevalente
organizzazione a maggiore distanza

Self-Organizing Networks e apprendimento (*adattamento!*)

Il sistema è composto da parti, o componenti relativamente indipendenti, (*p.es. particelle, molecole, neuroni, cellule*) che interagiscono fra loro, stabilendo delle *connessioni*.

Non esiste un controllo centralizzato.

Il sistema riceve degli input

$$\mathbf{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \qquad (\mathbf{x}_i \in \mathbf{R}^d)$$

- ***Unsupervised learning***

  Il sistema si organizza in una configurazione che dipende da $\mathbf{D}$

  Il sistema si ***adatta***, nel senso che la configurazione che si ottiene dipende dalle caratteristiche intrinseche del sistema e dagli input $\mathbf{D}$

  Competitive learning

  Le componenti del sistema ***si differenziano***:
  ciascuna di esse dipende da un sottoinsieme di $\mathbf{D}$

  I sottoinsiemi sono mutuamente esclusivi.

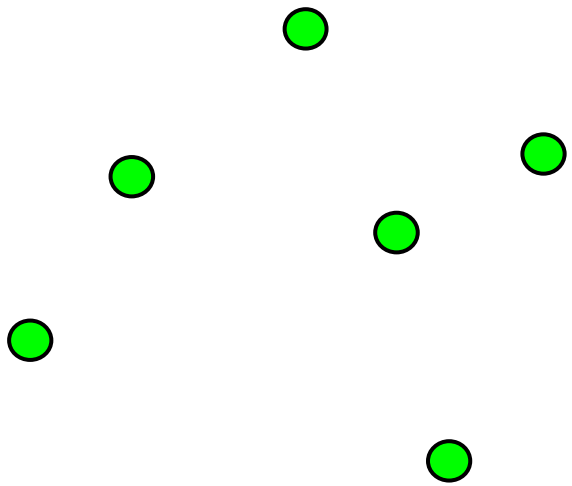
  Collaborative (reinforcement) learning

  Le componenti del sistema ***stabiliscono delle connessioni o interazioni***
  ciascuna delle quali è giustificata da un sottoinsieme di $\mathbf{D}$

Preludio topologico-geometrico

- **Tessellazione di Voronoi**

  I punti verdi sono anche detti *landmark*

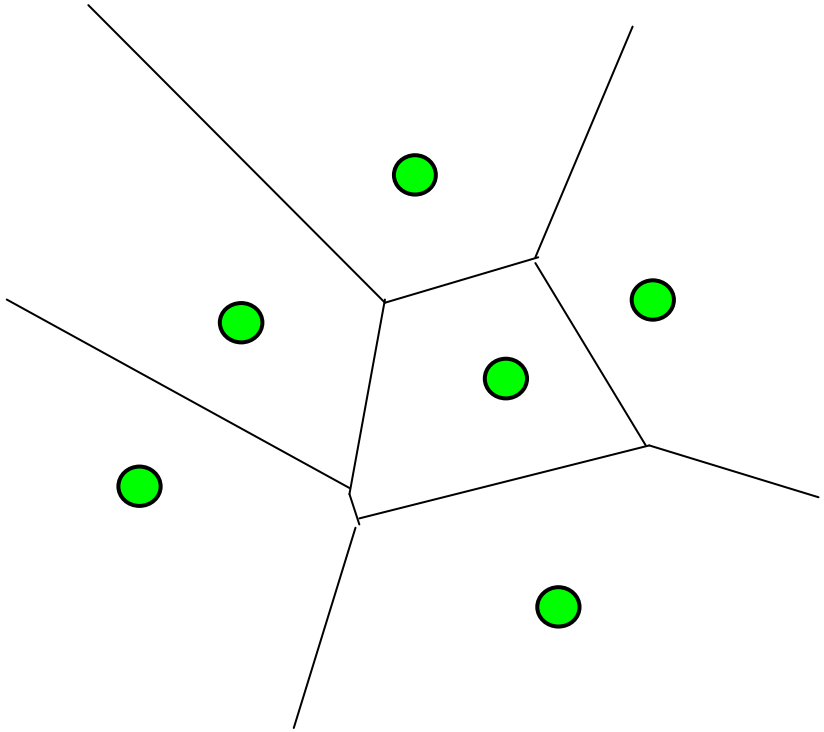


Preludio topologico-geometrico

- **Tessellazione di Voronoi**

  I punti verdi sono anche detti *landmark*

  La **regione di Voronoi** associata a ciascun *landmark* è formata da tutti i punti che sono più vicini al campione che a qualsiasi altro

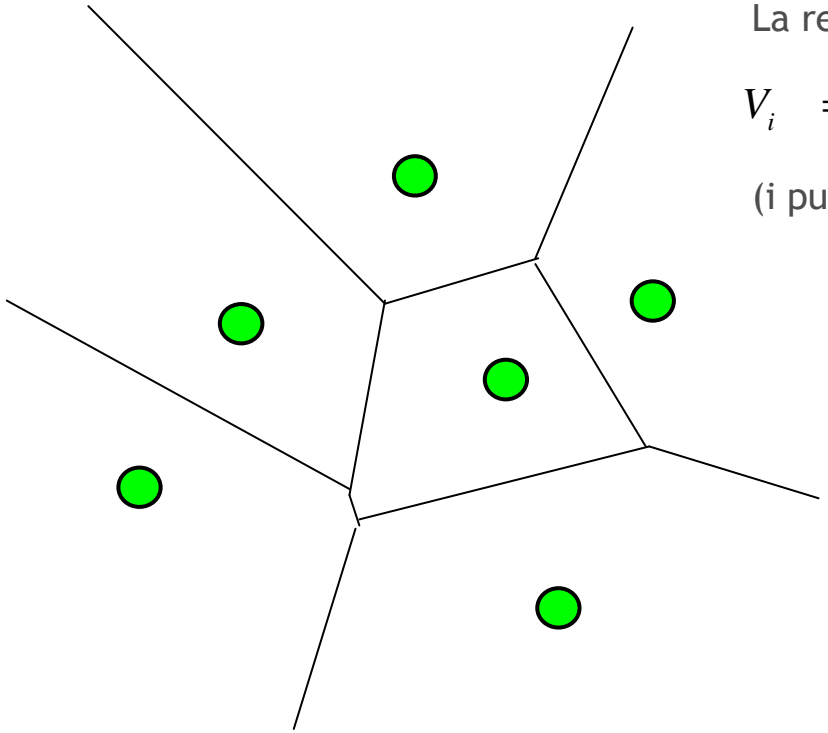


Preludio topologico-geometrico

- **Tessellazione di Voronoi**

  I punti verdi sono anche detti *landmark*

  La **regione di Voronoi** associata a ciascun *landmark* è formata da tutti i punti che sono più vicini al campione che a qualsiasi altro



La regione di Voronoi associata a ciascun $\boldsymbol{p}_i$

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

(i punti sul confine sono *condivisi*)

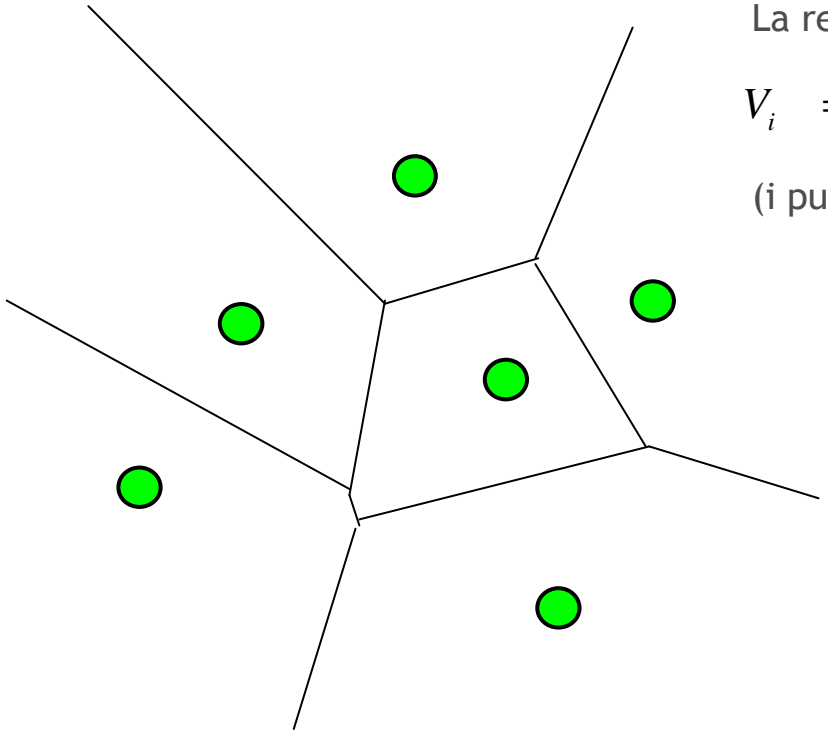
Preludio topologico-geometrico

- **Tessellazione di Voronoi**

  I punti verdi sono anche detti *landmark*

  La **regione di Voronoi** associata a ciascun *landmark* è formata da tutti i punti che sono più vicini al campione che a qualsiasi altro

  La **tessellazione di Voronoi** (di $\mathbf{R}^d$) è formata dall'insieme delle regioni di Voronoi



La regione di Voronoi associata a ciascun $\boldsymbol{p}_i$

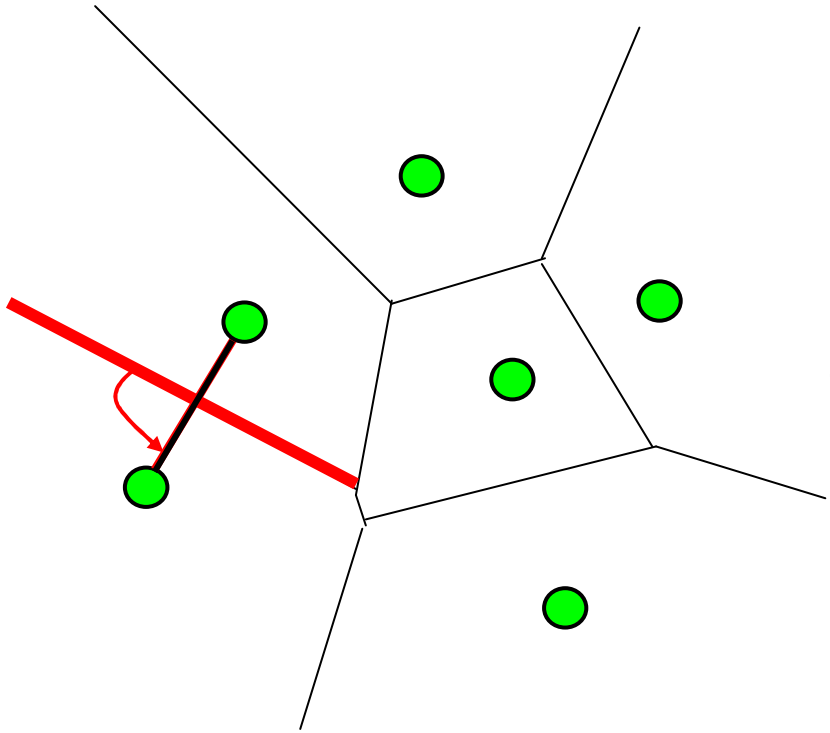
$$V_i = \left\{ \mathbf{x} \in \mathbf{R}^d \;\middle|\; i = \arg\min_j \left\| \mathbf{x} - \mathbf{p}_j \right\| \right\}$$

(i punti sul confine sono *condivisi*)

Preludio topologico-geometrico

- **Grafo di Delaunay**

  Due campioni le cui regioni di Voronoi hanno punti in comune (= sono confinanti) sono uniti da un arco



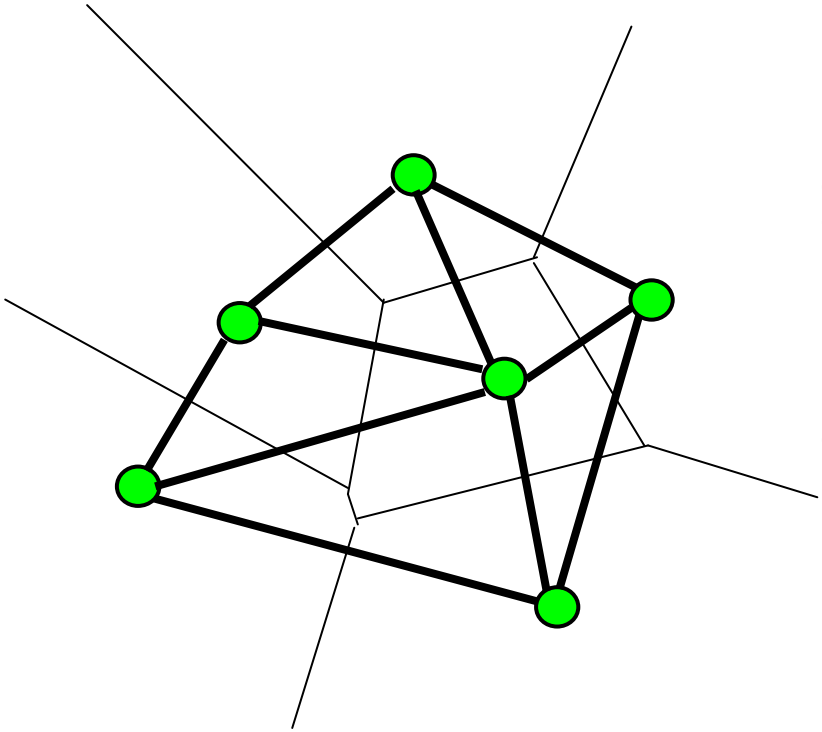
L'arco del grafo è ortogonale al confine tra le due regioni di Voronoi

Preludio topologico-geometrico

- **Grafo di Delaunay**

  Due *landmark* le cui regioni di Voronoi hanno punti in comune (= sono confinanti) sono uniti da un arco

  Il **grafo di Delaunay** è formato dai campioni e l'insieme di tutti gli archi che uniscono regioni Voronoi confinanti



Il grafo di Delaunay forma un *inviluppo convesso* dei campioni, diviso all'interno in regioni triangolari

La triangolazione di Delaunay ha proprietà notevoli, p.es. massimizza l'angolo minimo dei triangoli

k-means (*Generalized Lloyd's Algorithm - Competitive learning*)

- Modo *batch*

  Dato un set di dati $\mathbf{D}$ appartenenti a $\mathbf{R}^d$ (o comunque ad uno spazio metrico), si costruisce un insieme di k vettori (o *landmark*) $\mathbf{L}$ che *rappresenta* $\mathbf{D}$

  1) La posizione iniziale dei vettori di $\mathbf{L}$ viene scelta a caso in $\mathbf{R}^d$ (o in un sotto-spazio)
     $\mathbf{L} = \{ \mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k \}$
  2) Si calcola la regione di Voronoi $\mathbf{V}_k$ associata a ciascun vettore $\mathbf{w}_k$
  3) Si sposta ciascun $\mathbf{w}_k$ nel punto medio del sotto-insieme di $\mathbf{D}$ che appartiene a $\mathbf{V}_k$

  $$\mathbf{w}_k = \frac{1}{\left| \{\mathbf{x} \in \mathbf{V}_k\} \right|} \sum_{\{\mathbf{x} \in \mathbf{V}_k\}} \mathbf{x} \qquad (\mathbf{x} \in \mathbf{D})$$

  4) L'algoritmo termina se lo spostamento di ciascun $\mathbf{w}_k$ è inferiore a una data soglia

k-means (Generalized Lloyd's Algorithm - Competitive learning)

- Modo *batch*

  L'algoritmo ottimizza per *gradient descent* la funzione di errore:

$$E(\mathbf{D},\mathbf{L}) \overset{\text{def}}{=} \sum_i E(\mathbf{x}_i,\mathbf{L}) \overset{\text{def}}{=} \sum_i \frac{1}{2} \min_k \left\|\mathbf{x}_i - \mathbf{w}_k\right\|^2$$

  Notare che

$$\sum_i \frac{1}{2} \min_k \left\|\mathbf{x}_i - \mathbf{w}_k\right\|^2 = \sum_k \sum_{\mathbf{x}_i \in \mathbf{V}_k} \frac{1}{2} \left\|\mathbf{x}_i - \mathbf{w}_k\right\|^2 \overset{\text{def}}{=} \sum_k E(\mathbf{D},\mathbf{w}_k)$$

  Quindi, ad ogni iterazione e per ciascun $\mathbf{w}_k$ lo spostamento è

$$\Delta\mathbf{w}_k = -\varepsilon \frac{\partial E(\mathbf{D},\mathbf{w}_k)}{\partial \mathbf{w}_k} = \varepsilon \sum_{\mathbf{x}_i \in \mathbf{V}_k} (\mathbf{x}_i - \mathbf{w}_k)$$

  L'algoritmo iterativo converge per qualsiasi valore $0 < \varepsilon < 1$

  ε viene anche detta costante di apprendimento (*learning constant*)

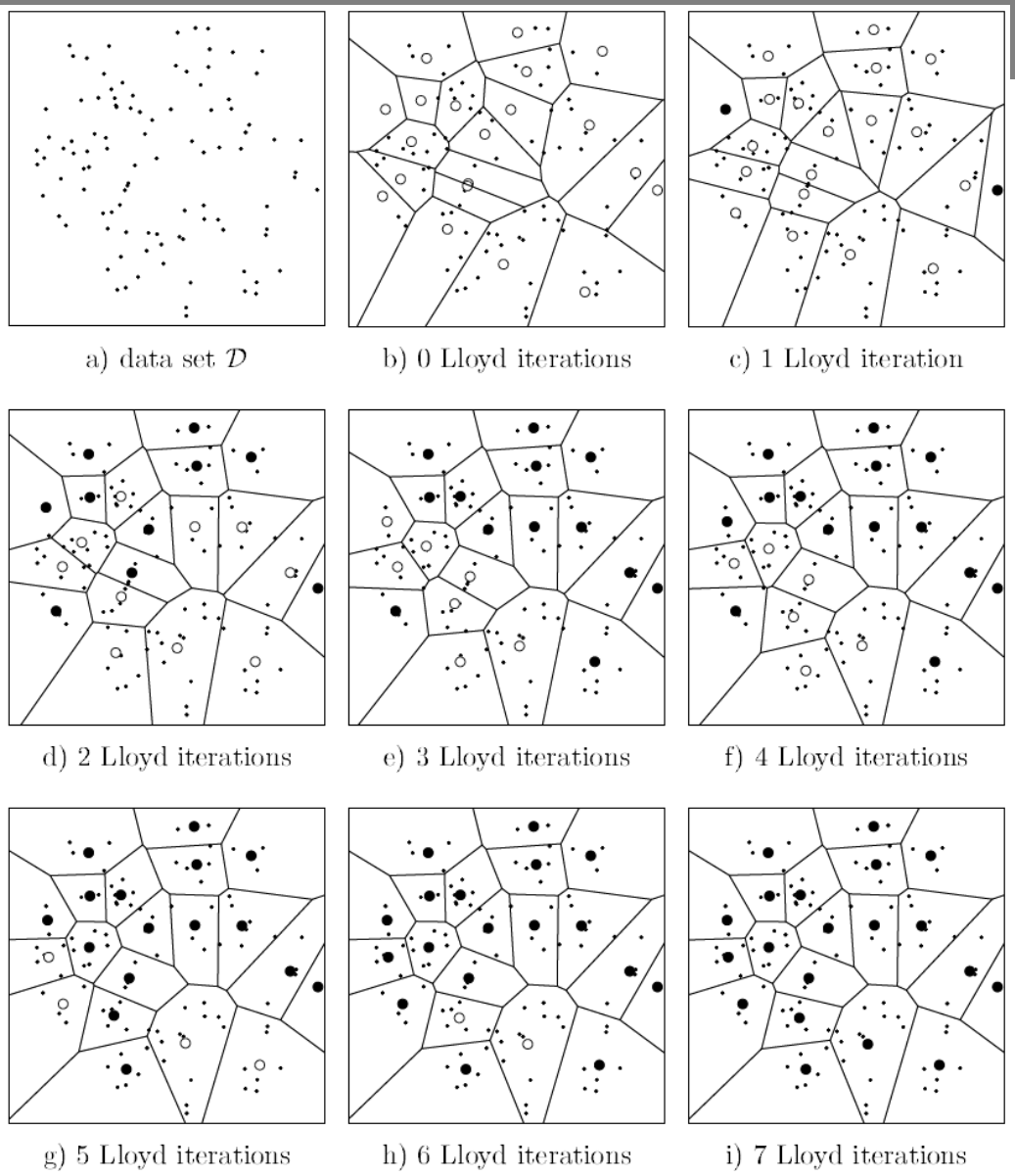
k-means

L'algoritmo di Lloyd in modo *batch*:
in ciascuna iterazione si considerano *tutti*
i vettori in **L** e i dati in **D**

L'insieme di vettori in **L** è talvolta chiamato
codebook : se si usa **L** per codificare **D**,
si ha un errore quadratico medio $E(\mathbf{D},\mathbf{L})$

Il minimo di $E(\mathbf{D},\mathbf{L})$ cui converge è locale,
quindi dipende dalla scelta iniziale di **L**

L'algoritmo adatta la tessellazione di Voronoi
indotta da **L** sul set di dati **D**



k-means e apprendimento probabilistico

- Il *k-means* come metodo di ottimizzazione

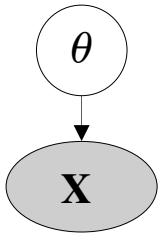
  Ottimizzazione iterativa ed alternata della funzione obiettivo:

  $$E(\mathbf{D},\mathbf{L}) = \sum_i \frac{1}{2} \min_k \left\| \mathbf{x}_i - \mathbf{w}_k \right\|^2 \qquad \propto \textit{Mean Square Error}$$

  a) Si minimizza $E(\mathbf{D}, \mathbf{L})$ rispetto a $\mathbf{D}$ assegnando i dati alla cella del $\mathbf{w}_k$ più vicino

  b) Si minimizza $E(\mathbf{D}, \mathbf{L})$ rispetto ai $\mathbf{L}$ spostando i *landmark* nel punto medio dei dati appartenenti al cluster

- Analogia: algoritmo *EM* (*Expectation-Maximization*)

  Modello grafico: n cluster, ciascuno con parametro $\{\mathbf{w}_k\}$:
  ciascun cluster genera i propri dati $\theta_i = k$ con distribuzione $P(\mathbf{X}, \theta \mid \{\mathbf{w}_k\})$

  Ottimizzazione iterativa ed alternata della funzione obbiettivo:

  $$MLE = \arg\max_{\{\mathbf{w}_k\}} P(\{\mathbf{X}_i\}, \{\theta_i\} \mid \{\mathbf{w}_k\}) \qquad \textit{Maximum Likelihood Estimator}$$

  a) Calcolo della distribuzione $P(\{\theta_i\} \mid \{\mathbf{X}_i\}, \{\mathbf{w}_k\})$

  b) *MLE* del <u>valor medio</u> di $L(\{\mathbf{w}_k\} \mid \{\mathbf{X}_i\}, \{\theta_i\})$, data la distribuzione $P(\{\theta_i\} \mid \{\mathbf{X}_i\}, \{\mathbf{w}_k\})$

  $$\{\mathbf{w}_k^*\} = \arg\max_{\{\mathbf{w}_k\}} \sum_{\{X_i\}} P(\{\theta_i\} \mid \{\mathbf{X}_i\}, \{\mathbf{w}_k\})\, P(\{\mathbf{X}_i\}, \{\theta_i\} \mid \{\mathbf{w}_k\})$$

k-means (*Generalized Lloyd's Algorithm - Competitive learning*)

- Modo *online*

  Dato un flusso di dati tratto da $\mathbf{D}$ appartenenti a $\mathbf{R}^d$,
  per esempio ottenuto per campionamento con probabilità $P(\mathbf{x})$,
  si costruisce un insieme di k vettori (o *landmark*) $\mathbf{L}$ che *rappresenta* $\mathbf{D}$

  1) La posizione iniziale dei vettori di $\mathbf{L}$ viene scelta a caso in $\mathbf{R}^d$
     (o in un sotto-spazio opportuno)
     $\mathbf{L} = \{ \mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k \}$
  2) Si riceve un input $\mathbf{x}$
  3) Si determina il vettore $\mathbf{w}_b$ più vicino a $\mathbf{x}$ (*winner*)

     $$b = \arg\min_k \| \mathbf{x} - \mathbf{w}_k \|$$

  4) Si sposta $\mathbf{w}_b$

     $$\Delta \mathbf{w}_b = \varepsilon (\mathbf{x} - \mathbf{w}_b)$$

  5) L'algoritmo termina se lo spostamento complessivo di ciascun $\mathbf{w}_k$
     durante un epoca (= per $|\mathbf{D}|$ input)è inferiore a una data soglia

k-means (Generalized Lloyd's Algorithm - Competitive learning)

- Modo *online*

  L'algoritmo ottimizza per *gradient descent* la funzione di errore:

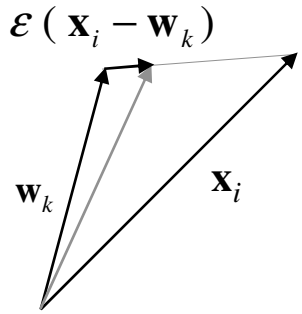
$$E(\mathbf{D},\mathbf{L}) \overset{\text{def}}{=} \sum_i E(\mathbf{x}_i,\mathbf{L}) \overset{\text{def}}{=} \sum_i \frac{1}{2} \min_k \left\|\mathbf{x}_i - \mathbf{w}_k\right\|^2$$

  Notare che

$$\sum_i E(\mathbf{D},\mathbf{L}) = \sum_k \sum_{\mathbf{x}_i \in \mathbf{V}_k} \frac{1}{2} \left\|\mathbf{x}_i - \mathbf{w}_k\right\|^2 \overset{\text{def}}{=} \sum_k \sum_{\mathbf{x}_i \in \mathbf{V}_k} E(\mathbf{x}_i,\mathbf{w}_k)$$

  Quindi, ad ogni input lo spostamento è

$$\Delta \mathbf{w}_k = -\varepsilon \frac{\partial E(\mathbf{x}_i,\mathbf{w}_k)}{\partial \mathbf{w}_k} = \varepsilon\,(\mathbf{x}_i - \mathbf{w}_k)$$



  L'algoritmo iterativo converge per qualsiasi valore $0 < \varepsilon < 1$

  ε viene anche detta costante di apprendimento (*learning constant*)

Hard Competitive Learning

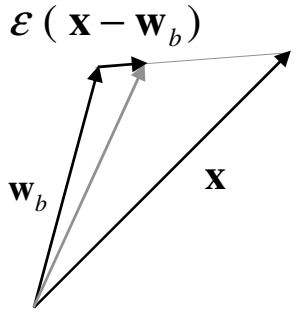
(anche *winner-takes-all*)

- Un algoritmo *incrementale*

  In questo caso non si ha un set di dati $\mathbf{D}$, ma un *flusso* di dati in input ed un insieme di n vettori $\mathbf{L}$ che si adatta.

  Si assume che i dati in input arrivino in sequenza, con probabilità $P(\mathbf{x})$

1) Ciascun vettore $\mathbf{w}_i$ viene scelto a caso in $\mathbf{R}^d$ (o in un sotto-spazio opportuno)
   $\mathbf{A} = \{ \mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n \}$
2) Generazione di un segnale $\mathbf{x}$ secondo $P(\mathbf{x})$
3) Si determina il *winner*
   $b = \arg\min_k \| \mathbf{x} - \mathbf{w}_k \|$
4) Si adatta il *winner* al segnale
   $\Delta \mathbf{w}_b = \varepsilon (\mathbf{x} - \mathbf{w}_b)$ ε è il *learning rate*

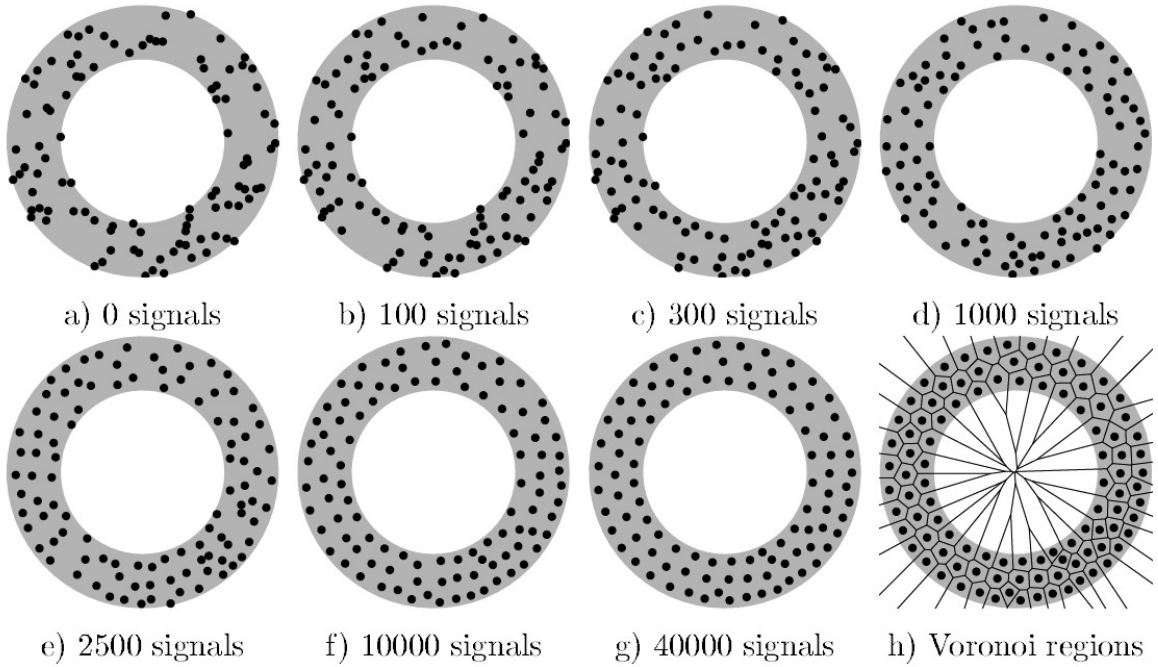


5) L'algoritmo termina se il flusso è esaurito, altrimenti torna a 2)

Hard Competitive Learning

A prima vista, il comportamento è simile al *k*-means

In realtà l'algoritmo NON CONVERGE: il *winner* si adatta sempre al segnale e quindi si sposta



Per forzare la convergenza, si può far 'raffreddare' (diminuire) il *learning rate* ε nel tempo

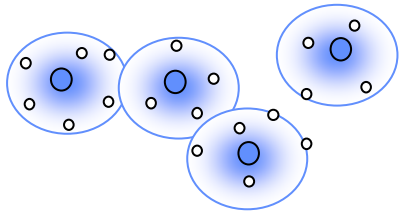
Con:

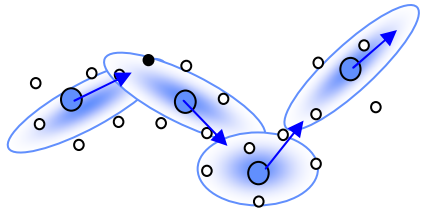
$$\varepsilon = \frac{1}{t}$$ (t numero di segnali)

Diversamente, ciascun $\mathbf{w}_i$ converge *asintotticamente* al valor medio di $P(\mathbf{x})$ nella regione di Voronoi $\mathbf{w}_i \rightarrow E(\mathbf{x} \mid \mathbf{x} \in \mathbf{V}_i)$

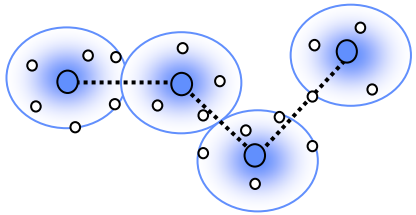
Vettori o reti? (competizione o collaborazione?)

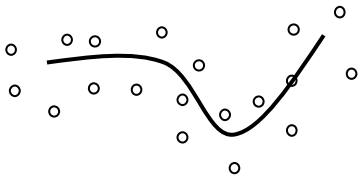
- Adattamento di vettori (*clustering*)
  - Ai vettori di A è associata una regione (di Voronoi)
  - Adattare un insieme di vettori A ad un flusso di dati in input significa adattare la tessellazione complessiva
  - Non si ha alcuna rappresentazione delle connessioni tra regioni diverse
    - Le connessioni che interessano sono quelle del flusso di dati (es. modello *hidden* composto da una curva con rumore)





- Struttura a rete
  - Formata da un'insieme di unità (nodi)
  - A ciascuna unità è associato un vettore
  - Le unità possono essere connesse
  - Adattare una rete significa adattare la tessellazione (indotta dai vettori) e le connessioni





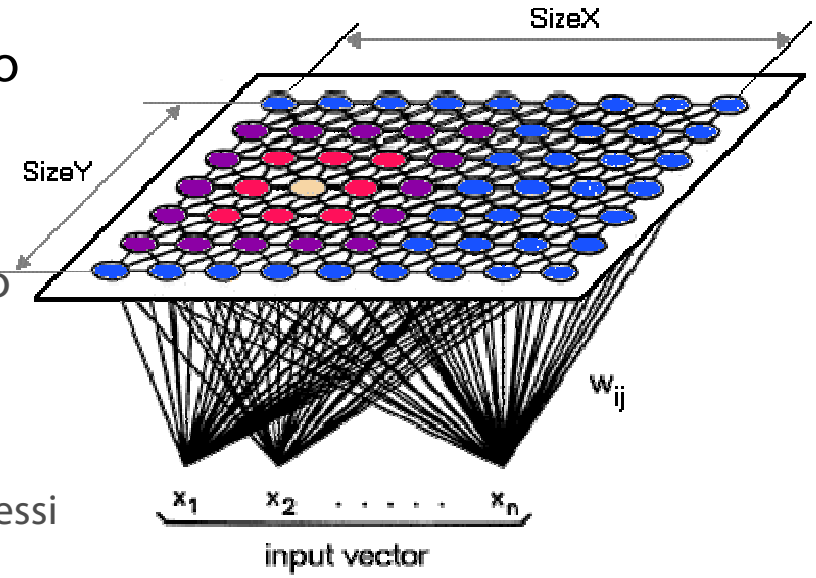
Self-Organizing Maps (SOM) (Kohonen, 1985)

- Struttura (tipica) a due livelli

Livello di mappa, con unità organizzate secondo una topologia prestabilita

A ciascuna unità c_i è associato un vettore $\mathbf{w}_i$

Le unità del livello di mappa sono connesse tra loro



Livello di input, o ingressi

Il livello di mappa è completamente connesso agli ingressi (non sono queste le connessioni che ci interessano)

La struttura delle connessioni tra le unità al livello di mappa è prestabilita e non varia nel corso dell'adattamento

Ad esempio, potrebbe trattarsi di un reticolo a base quadrata

Self-Organizing Maps (SOM)

- Adattamento incrementale

  Un insieme di unità $\mathbf{A} = \{ c_1, c_2, \ldots, c_n \}$ a ciascuna delle quali è associato un vettore $\mathbf{w}_i \in R^d$

  Un insieme di connessioni $\mathbf{C} : \mathbf{A} \times \mathbf{A}$ prefissato (p.es.un reticolo quadrato)

  Un flusso di dati $\mathbf{x} \in \mathbf{R}^d$ in input, con probabilità $P(\mathbf{x})$

  1) Ciascun vettore $\mathbf{w}_i$ viene scelto a caso in $\mathbf{R}^d$
  2) Generazione di un segnale $\mathbf{x}$ secondo $P(\mathbf{x})$
  3) Si determina il *winner*

  $$b = \arg\min_k \| \mathbf{x} - \mathbf{w}_k \|$$

  4) Si adattano *tutti* i vettori di $\mathbf{A}$ al segnale

  $$\Delta \mathbf{w}_i = \varepsilon(t)\, h(i,s)\, (\mathbf{x} - \mathbf{w}_i)$$

  $\varepsilon(t)$ è un *learning rate* che varia nel tempo

  $h(i,s)$ è una funzione che varia con la *distanza* nella rete tra c_i e c_b

  5) L'algoritmo termina se il flusso è esaurito, altrimenti torna a 2)

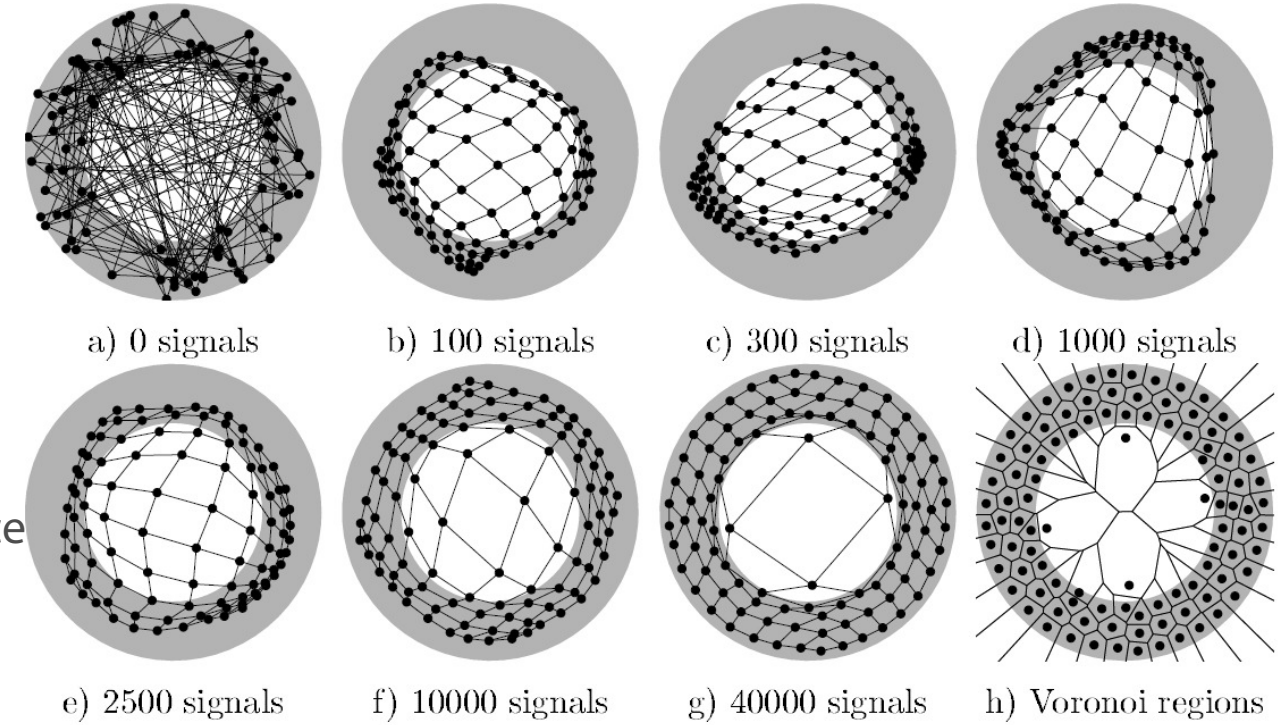
Self-Organizing Maps (SOM)

- Scelta delle funzioni

  Learning rate variabile

  $$\varepsilon(t) = \varepsilon_{\max} \, (\varepsilon_{\min} / \varepsilon_{\max})^{\, t/t_{\max}}$$

  $\varepsilon(t)$ decresce da un valore massimo ad uno minimo



  Adattamento delle unità

  Una gaussiana che definisce un *intorno* del *winner*

  $$h(i,s) = \exp\left(\frac{d(c_i, c_s)}{2\sigma(t)^2}\right)$$

  avente una varianza $\sigma(t)$ decresce nel tempo: $\sigma(t) = \sigma_{\max} \, (\sigma_{\min} / \sigma_{\max})^{\, t/t_{\max}}$

  dove $d(c_i, c_s)$ è la distanza tra le unità nel reticolo (p.es. una *manhattan distance*)

  In pratica, l'*intorno* del *winner* si restringe nel tempo

Self-Organizing Maps (SOM)

- SOM come metodo di ottimizzazione

  Non è descrivibile nel modo 'classico': è un'ottimizzazione distribuita

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

  Non esiste una funzione obiettivo complessiva che viene ottimizzata da una SOM

  Esiste una funzione obiettivo che ciascun nodo ottimizza in modo indipendente

Hebbian learning e grafo di Delaunay (Martinetz e Schulten, 1993)

- *Collaborative learning* per una costruzione incrementale

Un insieme di unità $\mathbf{A} = \{ c_1, c_2, \ldots, c_n \}$ a ciascuna delle quali è associato un vettore $\mathbf{w}_i \in \mathbf{R}^d$

Un insieme di connessioni $\mathbf{C}$ inizialmente vuoto

Un flusso di dati $\mathbf{x} \in \mathbf{R}^d$ in input, con probabilità $P(\mathbf{x})$

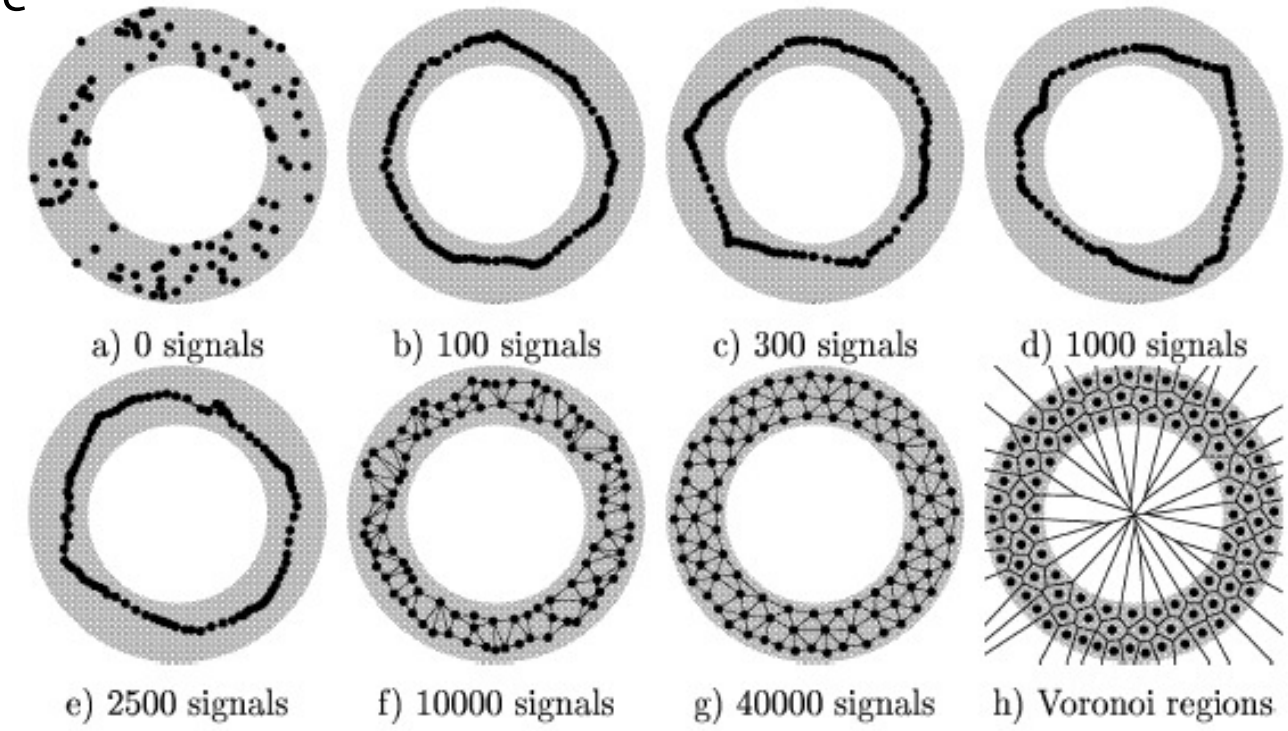
1) I vettori $\mathbf{w}_i$ vengono scelti a caso in $\mathbf{R}^d$
2) Generazione di un segnale $\mathbf{x}$ secondo $P(\mathbf{x})$
3) Si determina il *winner* ed il *second winner*

$$b = \arg\min_k \left\| \mathbf{x} - \mathbf{w}_k \right\| \qquad s = \arg\min_{k \neq b} \left\| \boldsymbol{\xi} - \mathbf{w}_k \right\|$$

3) Si aggiunge a $\mathbf{C}$ la connessione (c_b, c_s), se non esiste già
4) L'algoritmo termina se il flusso è esaurito, altrimenti torna a 2)

Neural Gas con Hebbian learning (Martinetz e Schulten, 1991)

Un algoritmo incrementale per adattare le connessioni



Utilizza lo *Hebbian learning* per le connessioni

Utilizza un meccanismo di aging per eliminare connessioni e unità non più necessarie

Neural Gas con Hebbian learning

- Un algoritmo incrementale per adattare le connessioni

  Un insieme di unità $\mathbf{A} = \{ c_1, c_2, \ldots, c_n \}$ a ciascuna delle quali è associato un vettore $\mathbf{w}_i \in \mathbf{R}^d$

  Un insieme di connessioni $\mathbf{C}$ inizialmente vuoto

  Un flusso di dati $\mathbf{x} \in \mathbf{R}^d$ in input, con probabilità $P(\mathbf{x})$

  1) I vettori $\mathbf{w}_i$ vengono scelti a caso in $\mathbf{R}^d$
  2) Generazione di un segnale $\mathbf{x}$ secondo $P(\mathbf{x})$
  3) Si determina il *winner* ed il *second winner*

  $$b = \arg\min_k \| \mathbf{x} - \mathbf{w}_k \| \qquad s = \arg\min_{k \neq b} \| \xi - \mathbf{w}_k \|$$

  4) Si aggiunge a $\mathbf{C}$ la connessione (c_b, c_s), se non esiste già
     Alla connessione (c_b, c_s) viene assegnata $age = 0$
  5) Si adattano <u>*tutti*</u> i vettori di $\mathbf{A}$ al segnale

  $$\Delta \mathbf{w}_i = \varepsilon(t)\, h(i,s)\, (\mathbf{x} - \mathbf{w}_i)$$

  $\varepsilon(t)$ è un *learning rate* che varia nel tempo

  $h(i,s)$ è una funzione che varia con *ranking* tra le unità c_i in base alla <u>*distanza*</u> da c_b

(*continua*)

Neural Gas con Hebbian learning

6) Si incrementa il valore di age di tutte le connessioni dell'unità *winner*

$$age_{(c_i, c_b)} = age_{(c_i, c_b)} + 1 \ , \ i \,\big|\, (c_i, c_b) \in \mathbf{C}$$

7) Si rimuovono tutte le connessioni con valore di age maggiore di una soglia.

8) L'algoritmo termina se il flusso è esaurito, altrimenti torna a 2)

Neural Gas con Hebbian learning

- NG come metodo di ottimizzazione

  Ottimizzazione per *gradient descent* stocastico della funzione

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

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

- Adattamento medio delle unità in un NG

Distribuzione di probabilità dei segnali di input

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

Densità (spaziale) delle unità

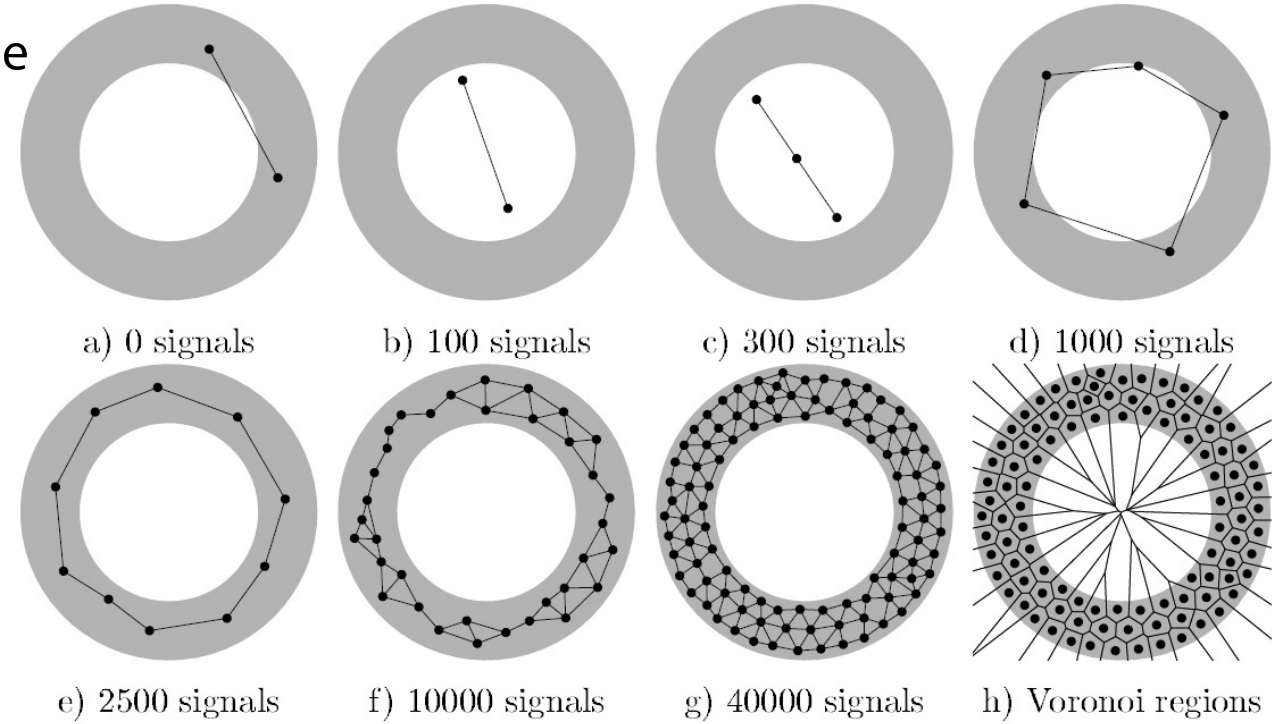
Ciascuna unità è soggetta a due 'forze':

1) Una che 'tira' verso le zone a maggiore densità di segnale
2) Un'altra che respinge le unità

   Con una distribuzione P uniforme, le unità si dispongono con densità uniforme (massima entropia)

Growing Neural Gas (Fritzke, 1995)

Un algoritmo incrementale per adattare unità e connessioni



Utilizza lo *Hebbian learning* per le connessioni

Introduce nuove unità in base all'*errore accumulato*

Utilizza un meccanismo di aging per eliminare connessioni e unità non più necessarie

Growing Neural Gas (GNG)

- Un algoritmo incrementale per adattare unità e connessioni

  Un insieme che inizialmente contiene solo due unità $\boldsymbol{A} = \{ c_1, c_2 \}$
  a ciascuna delle quali è associato un vettore $\mathbf{w}_i \in \mathbf{R}^d$

  Un insieme di connessioni $\mathbf{C}$ inizialmente vuoto

  Un flusso di dati $\mathbf{x} \in \mathbf{R}^d$ in input, con probabilità $P(\mathbf{x})$

  1) Vengono create due unità c_1 e c_2 con due vettori $\mathbf{w}_1$ e $\mathbf{w}_2$ scelti a caso in $\mathbf{R}^d$
  2) Generazione di un segnale $\mathbf{x}$ secondo $P(\mathbf{x})$
  3) Si determina il *winner* ed il *second winner*

$$b = \arg\min_k \| \mathbf{x} - \mathbf{w}_k \| \qquad s = \arg\min_{k \neq b} \| \xi - \mathbf{w}_k \|$$

  4) Si aggiunge a $\mathbf{C}$ la connessione (c_b, c_s), se non esiste già
     Alla connessione (c_b, c_s) viene assegnata $age = 0$
  5) Si incrementa l'errore accumulato dall'unità *winner*

$$\Delta E_b = \| \mathbf{x} - \mathbf{w}_b \|^2$$

  6) Si adattano i vettori dell'unità *winner* e delle unità direttamente connesse

$$\Delta \mathbf{w}_b = \varepsilon_b (\mathbf{x} - \mathbf{w}_b) \qquad \Delta \mathbf{w}_i = \varepsilon_n (\xi - \mathbf{w}_i), \; i \mid (c_i, c_b) \in C$$

(*continua*)

Growing Neural Gas

7) Si incrementa il valore di age di tutte le connessioni dell'unità *winner*

$$age_{(c_i, c_b)} = age_{(c_i, c_b)} + 1 \ , \ i \,|\, (c_i, c_b) \in \mathbf{C}$$

8) Si rimuovono tutte le connessioni con valore di age maggiore di una soglia. Vengono rimosse anche le unità che rimangono prive di connessioni.

9) Se il numero di segnali generati è multiplo di un valore prestabilito, si inserisce una nuova unità

9.1) Si individua l'unita c_q con il massimo errore accumulato E_q e l'unità connessa c_f con il valore di E più elevato

9.2) Si rimuove da $\mathbf{C}$ la connessione (c_q, c_f)

9.3) Si aggiunge ad $\mathbf{A}$ una nuova unità c_r con un vettore associato che è la media dei due

$$\mathbf{w}_r = (\mathbf{w}_q + \mathbf{w}_f) / 2$$

9.4) Si aggiungono a $\mathbf{C}$ le connessioni (c_q, c_r) e (c_r, c_f) , con $age = 0$

9.5) Si ripartiscono i valori di E tra c_q , c_f e c_r

10) Si decrementano tutti i valori di errore accumulato

$$\Delta E_i = -\beta E_i$$

11) L'algoritmo termina se il flusso è esaurito, altrimenti torna a 1)

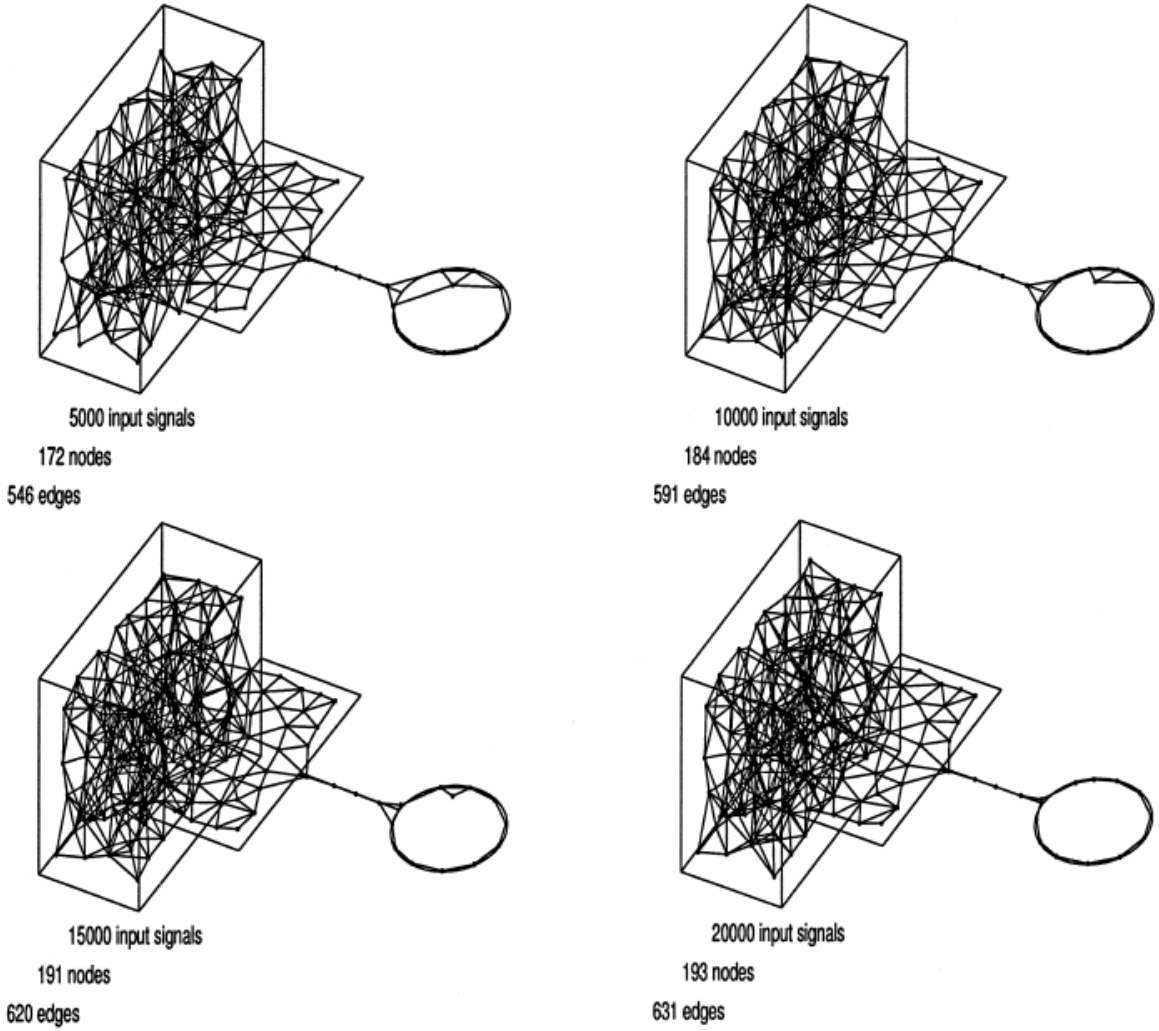
Adattamento e topologia variabile

Rispetto alle *SOM* (topologia fissa) i *GNG* possono adattare la propria topologia a quella del flusso di dati

Qui a destra: il flusso di dati proviene da una struttura composta da parti a dimensione intrinseca diversa

La rete si adatta e l'organizzazione risultante rappresenta la topologia *della struttura originaria*

Il calcolo è distribuito: ogni unità 'vede' soltanto i propri vicini



Grow-when-required networks (Marsland, 2002)

- Un limite dei *Growing Neural Gas*

  La rete cresce aggiungendo un'unità a periodi stabiliti

  La crescita non ha limiti, salvo ovviamente un massimo prestabilito

  Ciò rappresenta un'ulteriore elemento di perturbazione

- *Grow-when-required networks* (GWR)

  Molto simile ai *GNG*

  Due differenze principali:

  a) Le unità hanno un valore di *habituation* (assuefazione) che decresce esponenzialmente, con costante τ, tutte le volte che l'unità è *winner*

     Sotto una certa soglia di *habituation*, l'unità riduce la 'propensione all'adattamento'

  b) Sotto la soglia di *habituation*, ciascuna unità ha un raggio prestabilito di *activity* (che definisce una d-palla in $\boldsymbol{R}^d$)

     I segnali in input per cui l'unità è *winner* vengono accettati solo se cadono all'interno del raggio di *activity*

     In caso contrario, si crea una nuova unità