Intelligenza Artificiale II

Self-Organizing Systems

Introduzione

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.

  Self-organizing systems typically (though not always) display ***emergent properties***."

Emergence

- (Corning P.A., 2002, in *Wikipedia*)

  " Perhaps the most elaborate recent definition of **emergence** was provided by Jeffrey Goldstein in the inaugural issue of "Emergence". (Goldstein 1999)

  To Goldstein, emergence refers to "the arising of novel and coherent structures, patterns and properties during the process of self-organization in complex systems."

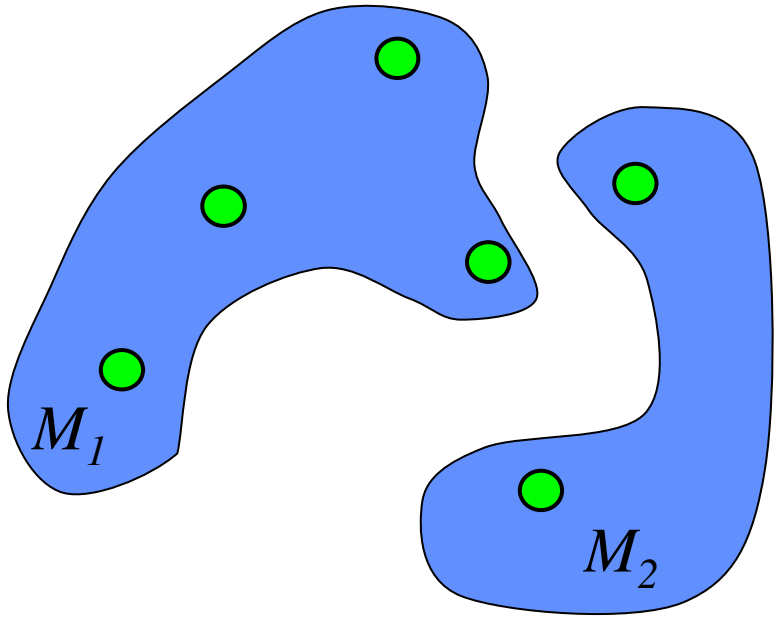
  The common characteristics are:

  1) radical novelty (features not previously observed in systems)
  2) coherence or correlation (meaning integrated wholes that maintain themselves over some period of time)
  3) a global or macro "level" (i.e. there is some property of "wholeness")
  4) it is the product of a dynamical process (it evolves)
  5) it is "ostensive" - it can be perceived "

Preludio topologico-geometrico (1)

- Tessellazione di Voronoi

  I punti verdi sono *campioni* tratti da due 'oggetti' M_1 e M_2

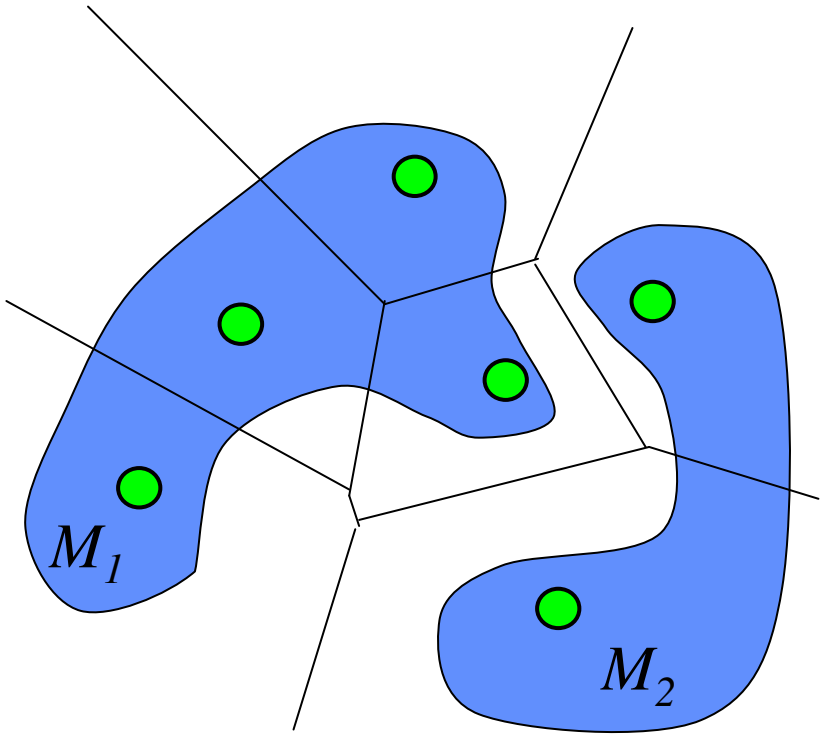


Preludio topologico-geometrico (1)

- Tessellazione di Voronoi

  I punti verdi sono *campioni* tratti da due 'oggetti' M_1 e M_2

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

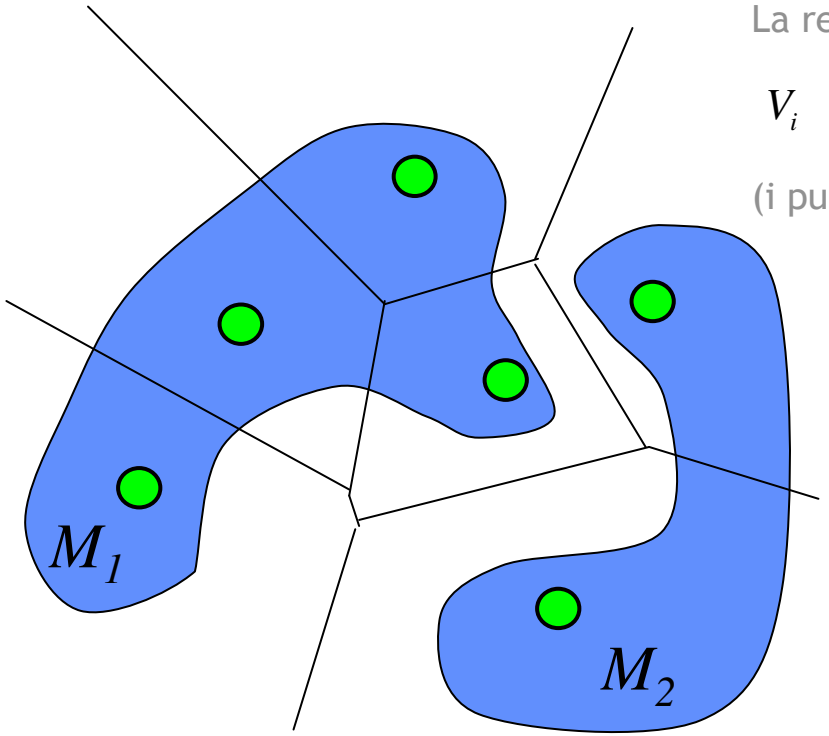


Preludio topologico-geometrico (1)

- **Tessellazione di Voronoi**

  I punti verdi sono *campioni* tratti da due 'oggetti' M_1 e M_2

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



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

$$V_i = \left\{ \boldsymbol{\xi} \in \mathrm{R}^d \,\middle|\, i = \arg\min_j \left\| \boldsymbol{\xi} - \boldsymbol{p}_j \right\| \right\}$$

(i punti sul confine sono *condivisi*)

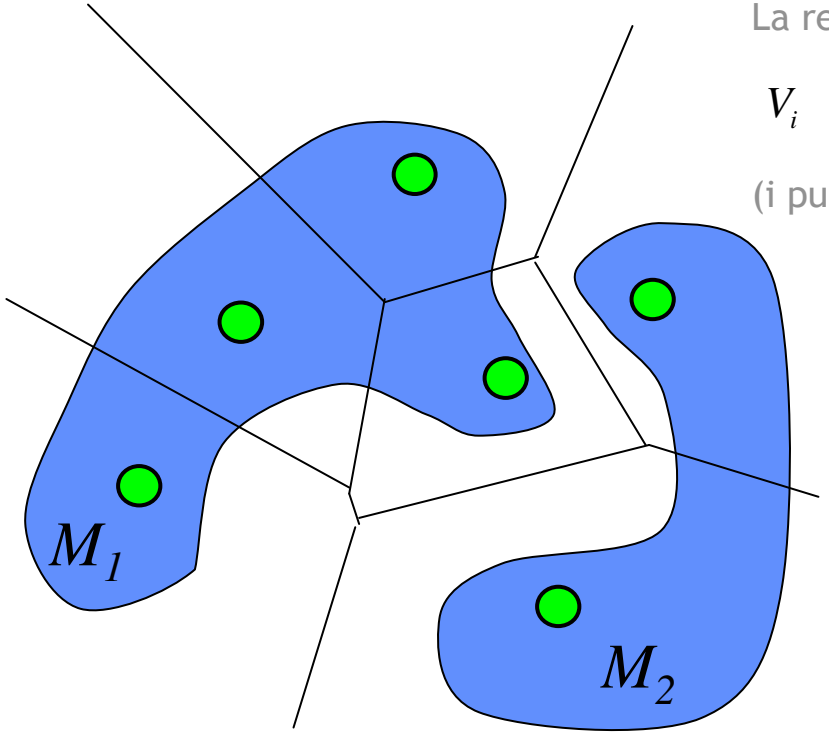
Preludio topologico-geometrico (1)

- **Tessellazione di Voronoi**

  I punti verdi sono *campioni* tratti da due 'oggetti' M_1 e M_2

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

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



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

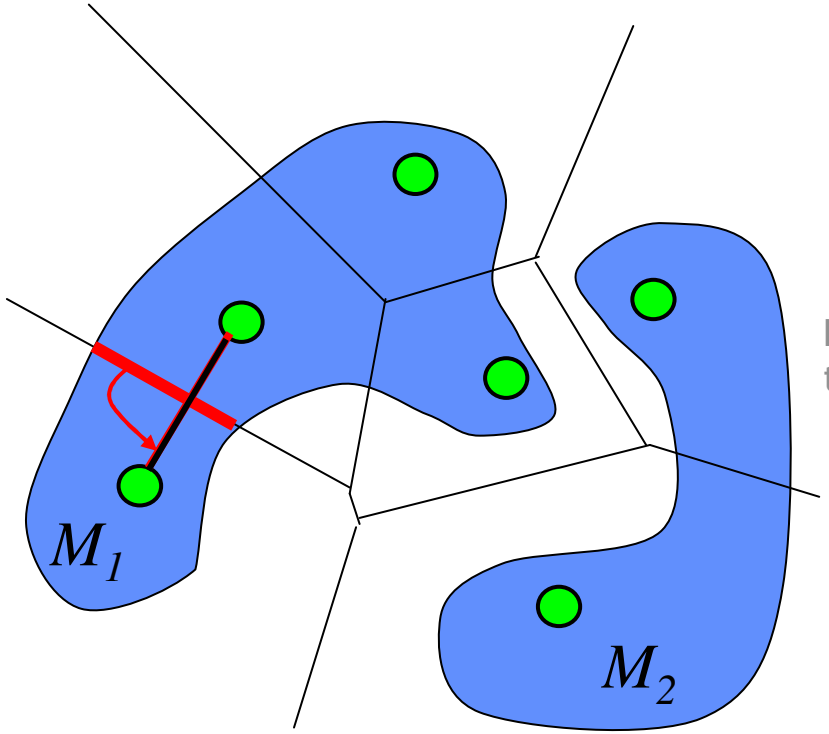
$$V_i = \left\{ \boldsymbol{\xi} \in \boldsymbol{R}^d \;\middle|\; i = \arg\min_j \left\| \boldsymbol{\xi} - \boldsymbol{p}_j \right\| \right\}$$

(i punti sul confine sono *condivisi*)

Preludio topologico-geometrico (2)

- Grafo di Delaunay

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



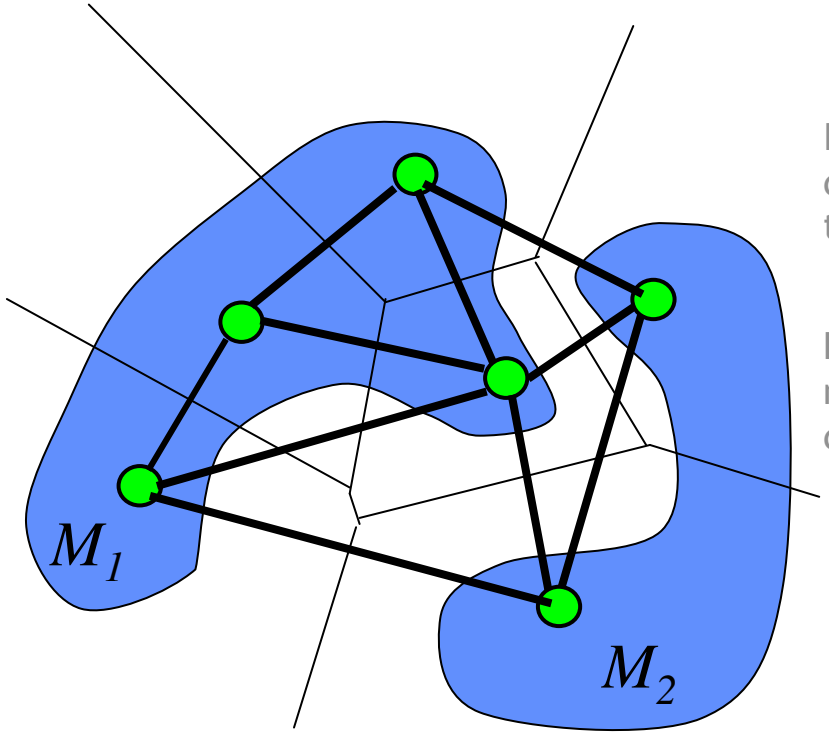
L'arco del grafo è ortogonale al confine tra le due regioni di Voronoi

Preludio topologico-geometrico (2)

- Grafo di Delaunay

  Due campioni 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*)

- Descrizione

Dato un set di dati $\boldsymbol{D}$ (appartenenti ad uno spazio metrico),
si costruisce un insieme di k vettori (o campioni) $\boldsymbol{A}$ che *rappresenta* $\boldsymbol{D}$

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

$$\boldsymbol{w}_i = \frac{1}{\left|\{\boldsymbol{\xi} \in \boldsymbol{V}_i\}\right|} \sum_{\{\boldsymbol{\xi} \in \boldsymbol{V}_i\}} \boldsymbol{\xi} \qquad (\boldsymbol{\xi} \in \boldsymbol{D})$$

4) L'algoritmo termina se nessun $\boldsymbol{w}_i$ è stato mosso, altrimenti torna a 2)

Dimostrabilmente, l'algoritmo converge ad un minimo *locale*
dell'errore quadratico medio:

$$MSE(\boldsymbol{D}, \boldsymbol{A}) = \frac{1}{|\boldsymbol{D}|} \sum_{\boldsymbol{w}_i \in \boldsymbol{A}} \sum_{\boldsymbol{\xi} \in \boldsymbol{V}_i} \left\| \boldsymbol{w}_i - \boldsymbol{\xi} \right\|^2$$

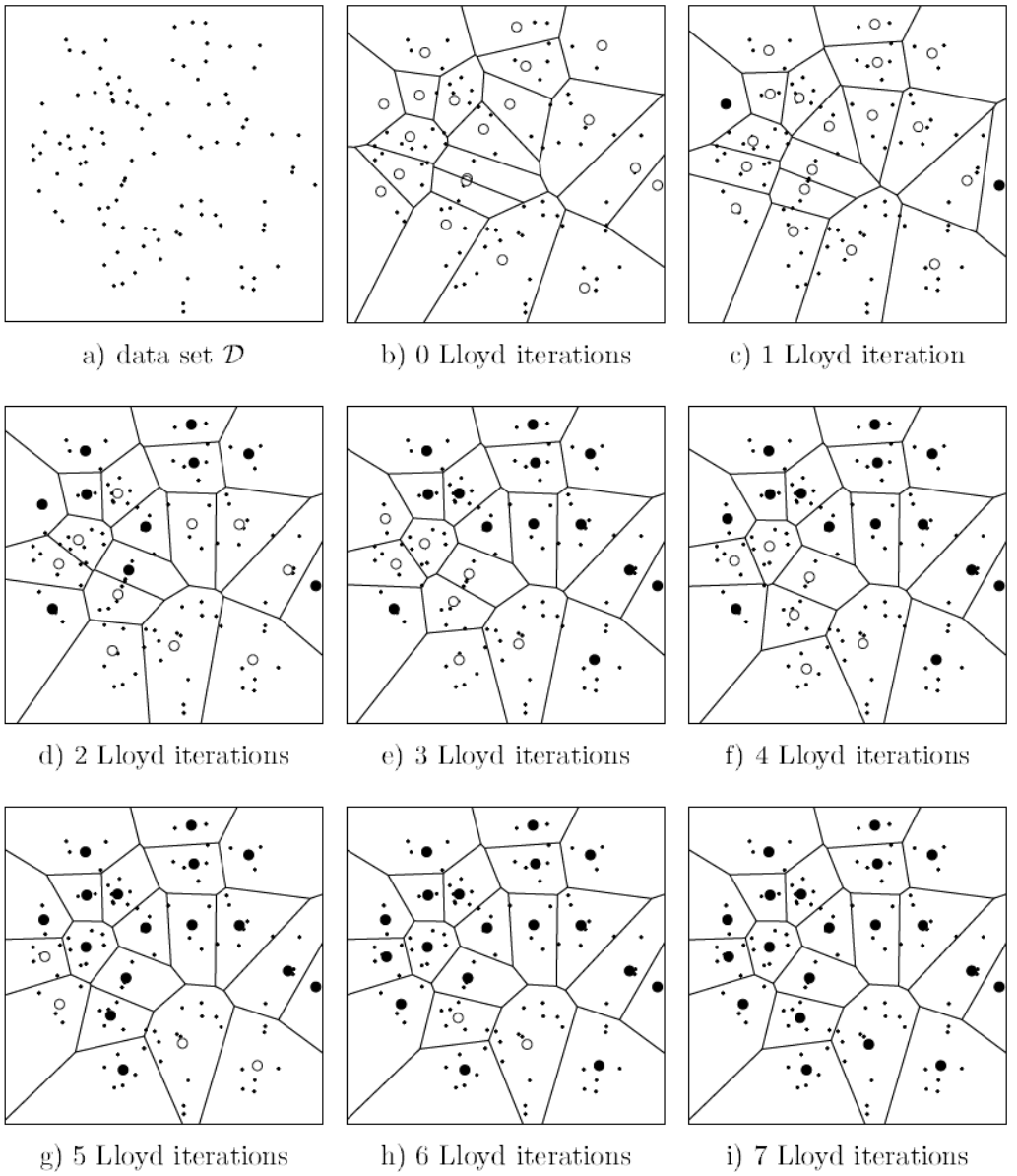
k-means

L'algoritmo di Lloyd opera in modo *batch*: in ciascuna iterazione si considerano *tutti* i vettori in $\boldsymbol{A}$ e i dati in $\boldsymbol{D}$

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

Il minimo di $MSE(\boldsymbol{D}, \boldsymbol{A})$ cui converge è locale, quindi dipende dalla scelta iniziale di $\boldsymbol{A}$

L'algoritmo adatta la tessellazione di Voronoi indotta da $\boldsymbol{A}$ al set di dati $\boldsymbol{D}$



k-means e apprendimento probabilistico

- Il *k-means* come metodo di ottimizzazione

  Ottimizzazione iterativa ed alternata della funzione obbiettivo:

  $$MSE(\boldsymbol{D},\boldsymbol{A}) = \frac{1}{|\boldsymbol{D}|}\sum_{\boldsymbol{w}_i \in \boldsymbol{A}}\sum_{\boldsymbol{\xi} \in \boldsymbol{V}_i} \|\boldsymbol{w}_i - \boldsymbol{\xi}\|^2$$ *Mean Square Error*

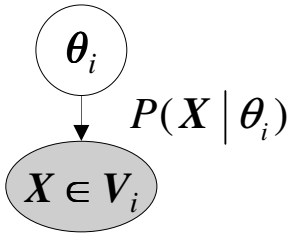
  a) Si minimizza $MSE(\boldsymbol{D}, \boldsymbol{A})$ rispetto a $\boldsymbol{\xi}$ assegnando i dati al cluster del $\boldsymbol{w}_i$ più vicino

  b) Si minimizza $MSE(\boldsymbol{D}, \boldsymbol{A})$ rispetto ai $\boldsymbol{w}_i$ spostandoli nel punto medio del cluster

  $$\boldsymbol{w}_i = E(\boldsymbol{\xi} \,|\, \boldsymbol{\xi} \in \boldsymbol{V}_i)$$

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

  Modello grafico: n cluster, ciascuno con parametri $\boldsymbol{\theta}_i$:
  ciascun cluster genera i propri dati $\boldsymbol{X} \in \boldsymbol{V}_i$ con distribuzione $P(\boldsymbol{X} \mid \boldsymbol{\theta}_i)$
  Sono disponibili solo le osservazioni $\boldsymbol{\xi} \in \boldsymbol{D}$



  Ottimizzazione iterativa ed alternata della funzione obbiettivo:

  $$MLE = P(\boldsymbol{D} \,|\, \boldsymbol{\theta})$$ *Maximum Likelihood Estimator*

  a) Calcolo della distribuzione $P(\boldsymbol{X} \mid \boldsymbol{\xi}, \{\boldsymbol{\theta}_i\})$

  b) *MLE* del <u>valor medio</u> di $L(\{\boldsymbol{\theta}_i\} \mid \boldsymbol{X}, \boldsymbol{\xi})$ rispetto a $\boldsymbol{X}$, data la distribuzione $P(\boldsymbol{X} \mid \boldsymbol{\xi}, \{\boldsymbol{\theta}_i\})$

  $$\{\boldsymbol{\theta}_i^*\} \;=\; \arg\max_{\{\boldsymbol{\theta}_i\}} \sum_{\{X_i\}} P(\boldsymbol{X} \mid \boldsymbol{\xi}, \{\boldsymbol{\theta}_i\})\, P(\boldsymbol{X}, \boldsymbol{\xi} \mid \{\boldsymbol{\theta}_i\})$$

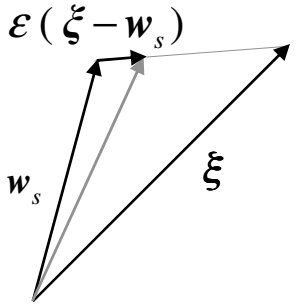
Hard Competitive Learning

(anche *winner-takes-all*)

- Un algoritmo *incrementale*

  In questo caso non si ha un set di dati $\boldsymbol{D}$, ma un *flusso* di dati in input
  un insieme di n vettori $\boldsymbol{A}$ che lo *rappresenta*
  Si assume che i dati in input arrivino in sequenza, con probabilità $P(\xi)$

1) Ciascun vettore $\boldsymbol{w}_i$ viene scelto a caso in $\boldsymbol{R}^d$ (o in un sotto-spazio opportuno)
   $\boldsymbol{A} = \{ \boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n \}$
2) Generazione di un segnale ξ secondo $P(\xi)$
3) Si determina il *winner*
   $$\boldsymbol{w}_s = \boldsymbol{w}_i \,\Big|\, i = \arg\min_j \left\| \boldsymbol{\xi} - \boldsymbol{w}_j \right\|$$
4) Si adatta il *winner* al segnale
   $$\Delta \boldsymbol{w}_s = \varepsilon (\boldsymbol{\xi} - \boldsymbol{w}_s)$$
   ε è il *learning rate*

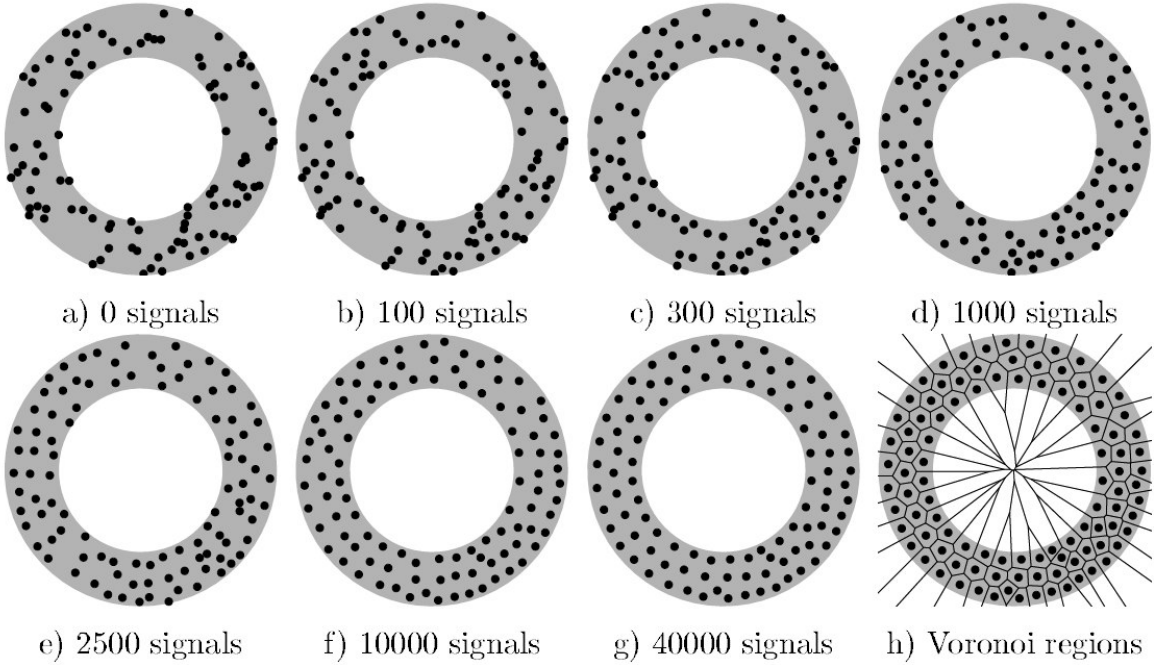


5) L'algoritmo termina se il flusso è esaurito, altrimenti torna a 2)

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:

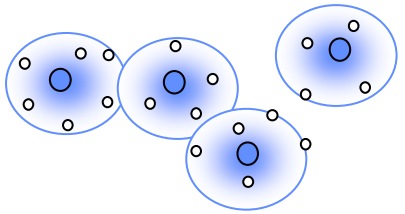
$$\varepsilon = \frac{1}{t}$$ (t numero di segnali)

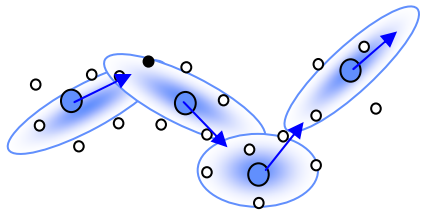
ciascun $\boldsymbol{w}_i$ converge *asintotticamente* al valor medio di $P(\boldsymbol{\xi})$ nella regione di Voronoi

$$\boldsymbol{w}_i \rightarrow E(\boldsymbol{\xi} \mid \boldsymbol{\xi} \in \boldsymbol{V}_i)$$

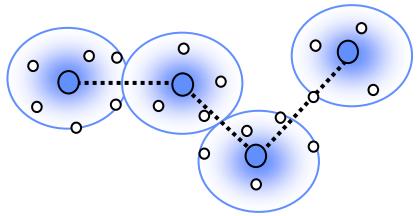
Adattare vettori o reti?

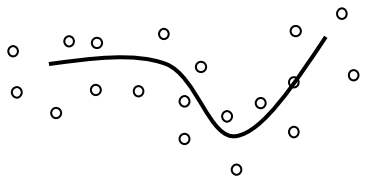
- Adattamento di vettori (*clustering*)
  - Ai vettori di A è associata una regione (di Voronoi)
  - Adattare un insieme di vettori A ad un flusso di dati in input significa adattare la tessellazione complessiva
  - Non si ha alcuna rappresentazione delle connessioni tra regioni diverse
    - Le connessioni 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

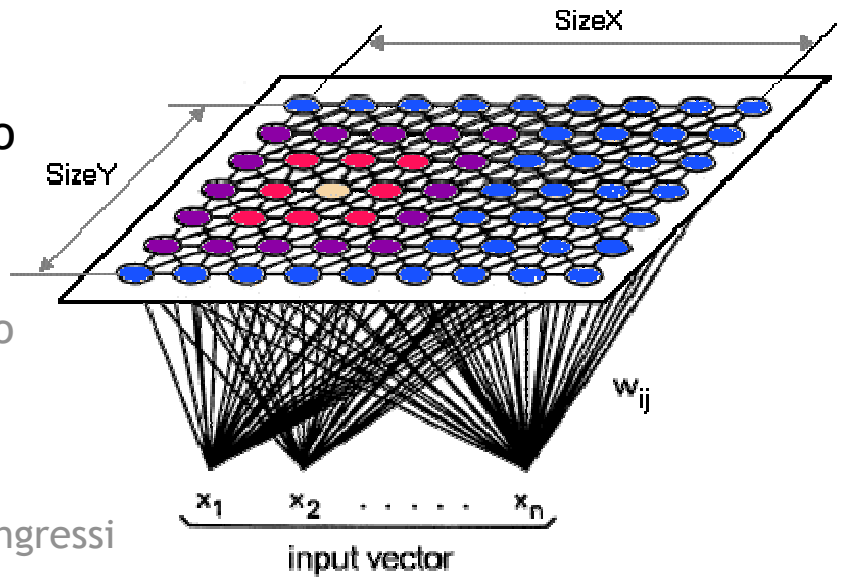
Self-organizing maps (*SOM*) (Kohonen, 1985)

- Struttura (tipica) a due livelli

  Livello di mappa, con unità organizzate secondo una topologia prestabilita

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

  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 $A = \{ c_1, c_2, \ldots, c_n \}$ a ciascuna delle quali è associato un vettore $w_i \in R^d$
  Un insieme di connessioni $C : A \times A$ prefissato (p.es.un reticolo quadrato)
  Un flusso di dati $\xi \in R^d$ in input, con probabilità $P(\xi)$

  1) Ciascun vettore w_i viene scelto a caso in R^d
  2) Generazione di un segnale ξ secondo $P(\xi)$
  3) Si determina il *winner*

     $$c_s = c_i \,\Big|\, i = \arg\min_j \left\| \xi - w_j \right\|$$

  4) Si adattano ***tutti*** i vettori di A al segnale

     $$\Delta w_i = \varepsilon(t) h(i,s)(\xi - w_i)$$

     $\varepsilon(t)$ 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_s
  5) L'algoritmo termina se il flusso è esaurito, altrimenti torna a 2)

Self-organizing maps (*SOM*)

- Scelta delle funzioni

  Learning rate variabile

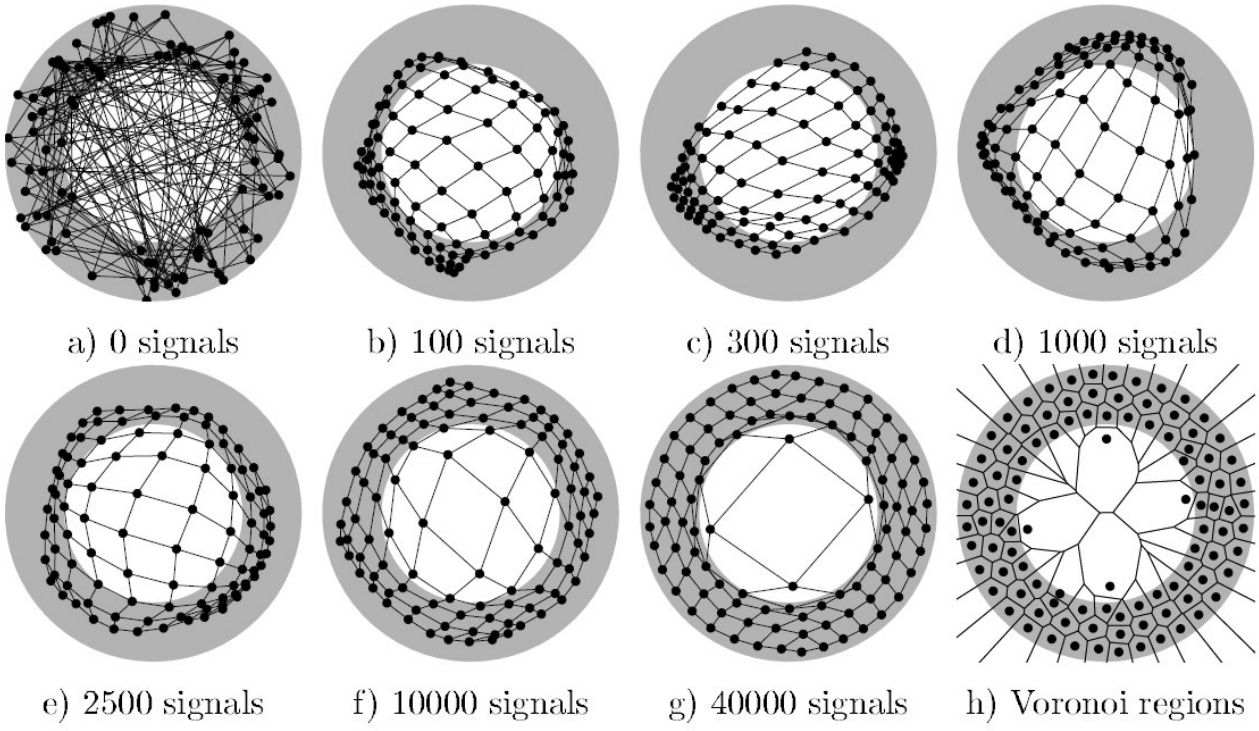
  $$\varepsilon(t) = \varepsilon_{\max} \, (\varepsilon_{\min} / \varepsilon_{\max})^{\, t/t_{\max}}$$

  $\varepsilon(t)$ 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

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

- Un algoritmo incrementale per costruire il grafo di Delaunay (o un suo sottoinsieme)

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

  Un insieme di connessioni C inizialmente vuoto

  Un flusso di dati $\xi \in R^d$ in input, con probabilità $P(\xi)$

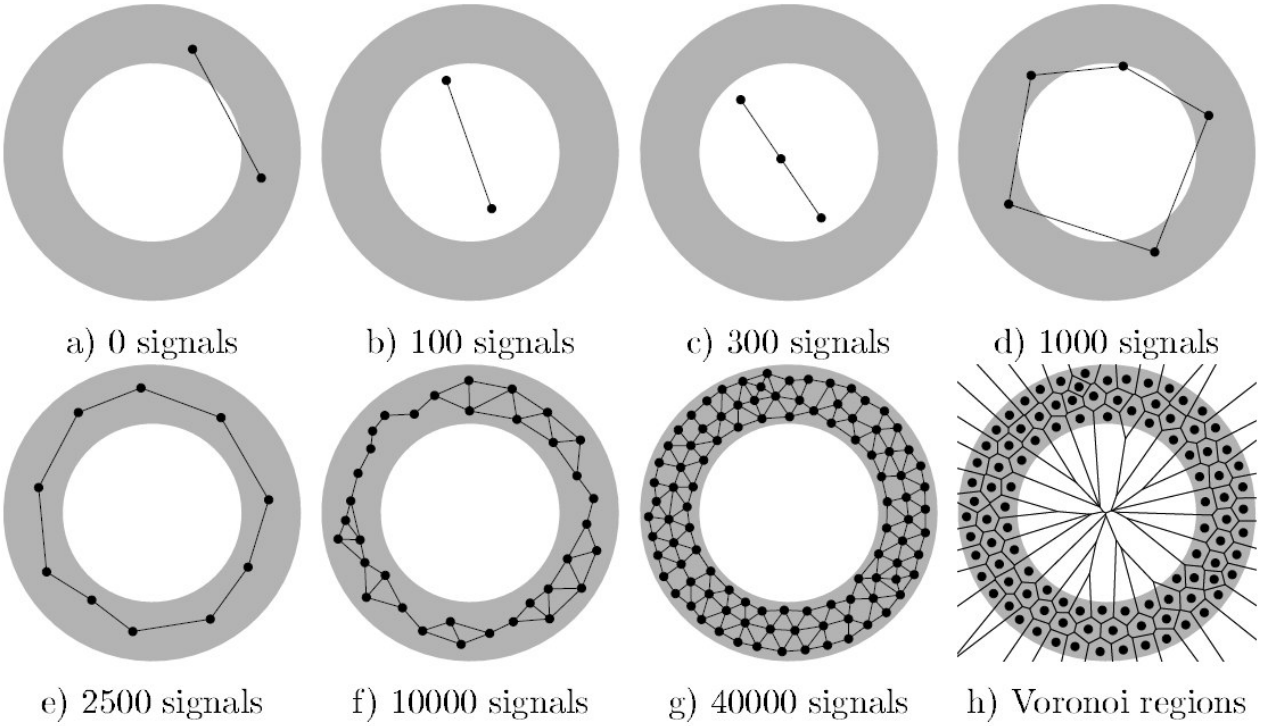
1) I due vettori $\boldsymbol{w}_i$ vengono scelti a caso in $\boldsymbol{R}^d$
2) Generazione di un segnale ξ secondo $P(\xi)$
3) Si determina il *winner* ed il *second winner*

$$c_s = c_i \,\Big|\, i = \arg\min_j \left\| \boldsymbol{\xi} - \boldsymbol{w}_j \right\| \qquad c_r = c_i \,\Big|\, i = \arg\min_{j \neq s} \left\| \boldsymbol{\xi} - \boldsymbol{w}_j \right\|$$

3) Si aggiunge a C la connessione (c_s, c_r), se non la contiene già
4) L'algoritmo termina se il flusso è esaurito, altrimenti torna a 2)

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 $A = \{ c_1, c_2 \}$
  a ciascuna delle quali è associato un vettore $\boldsymbol{w}_i \in R^d$

  Un insieme di connessioni C inizialmente vuoto

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

  1) I due vettori $\boldsymbol{w}_1$ e $\boldsymbol{w}_2$ vengono scelti a caso in $\boldsymbol{R}^d$
  2) Generazione di un segnale $\boldsymbol{\xi}$ secondo $P(\boldsymbol{\xi})$
  3) Si determina il *winner* ed il *second winner*

     $$c_s = c_i \,\Big|\, i = \arg\min_j \left\| \boldsymbol{\xi} - \boldsymbol{w}_j \right\| \qquad c_r = c_i \,\Big|\, i = \arg\min_{j \neq s} \left\| \boldsymbol{\xi} - \boldsymbol{w}_j \right\|$$

  4) Si aggiunge a C la connessione (c_s, c_r), se C non la contiene già
     Alla connessione (c_s, c_r) viene assegnata $age = 0$
  5) Si incrementa l'errore accumulato dall'unità *winner*

     $$\Delta E_s = \left\| \boldsymbol{\xi} - \boldsymbol{w}_s \right\|^2$$

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

     $$\Delta \boldsymbol{w}_s = \varepsilon_b (\boldsymbol{\xi} - \boldsymbol{w}_s) \qquad \Delta \boldsymbol{w}_i = \varepsilon_n (\boldsymbol{\xi} - \boldsymbol{w}_i), \; i \,\Big|\, (c_i, c_s) \in C$$

(*continua*)

Growing Neural Gas

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

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

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 C la connessione (c_q, c_f)

   9.3) Si aggiunge ad *A* una nuova unità c_r con un vettore associato che è la media dei due

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

   9.4) Si aggiungono a 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)

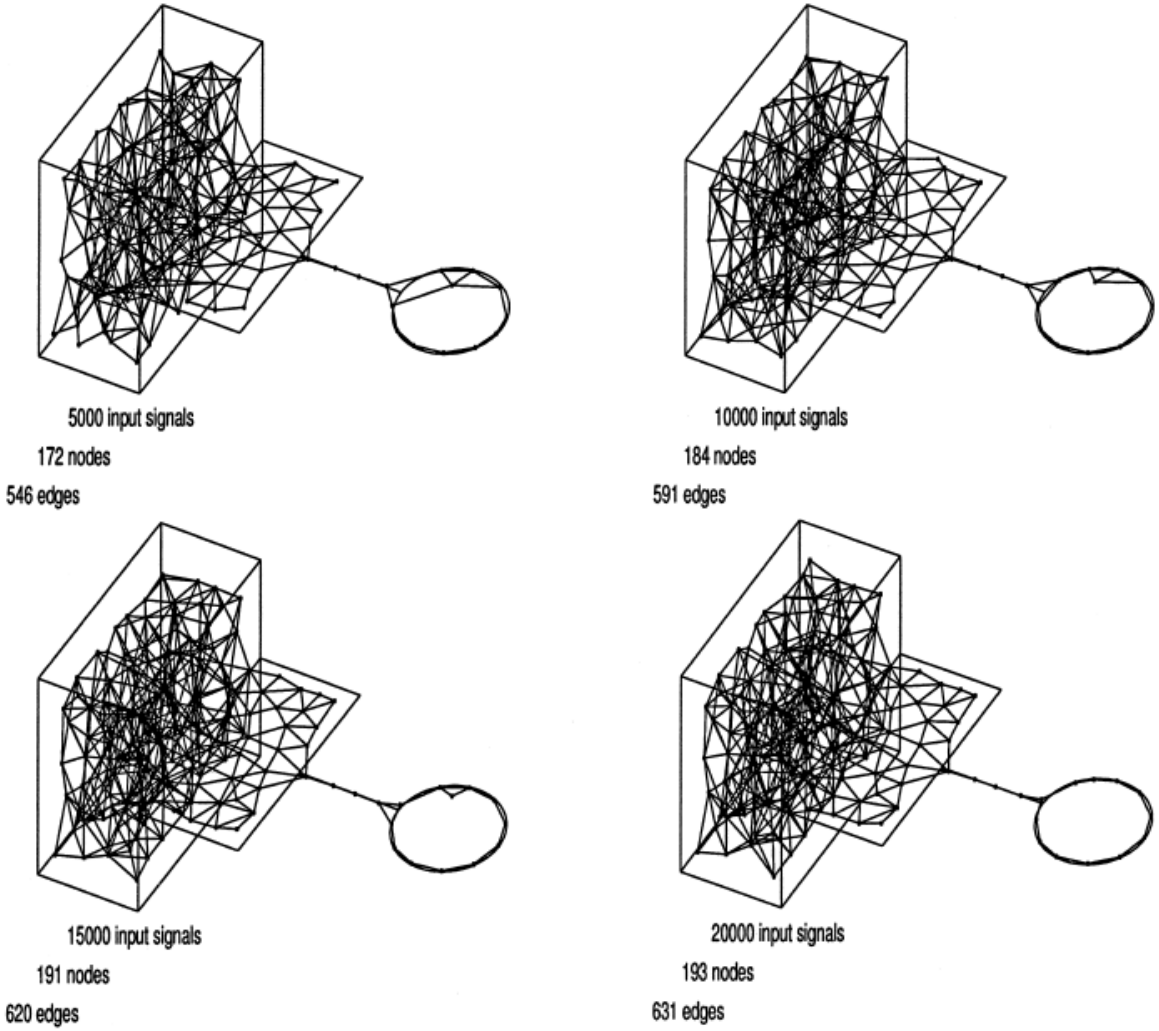
Adattamento e topologia variabile

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