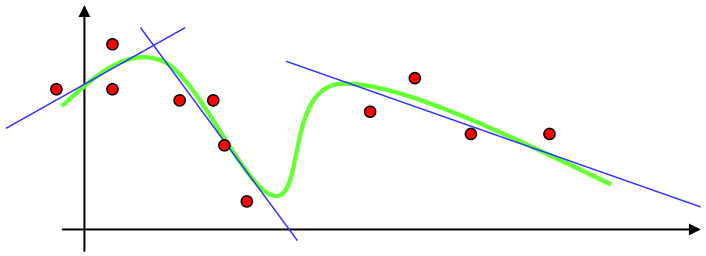
Local Methods

- Olive slides: Alpaydin
- Blue slides: Haykin, clarifications/notations

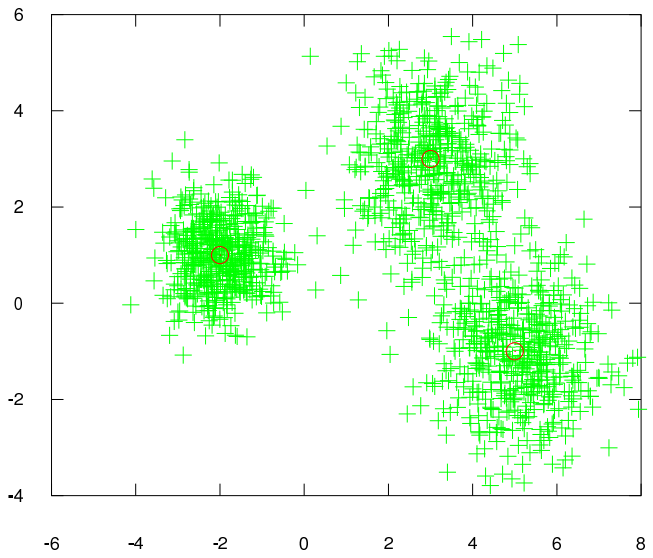
Introduction

- Divide the input space into local regions and learn simple (constant/linear) models in each patch



- Unsupervised: Competitive, online clustering
- Supervised: Radial-basis functions, mixture of experts

Competetive Learning

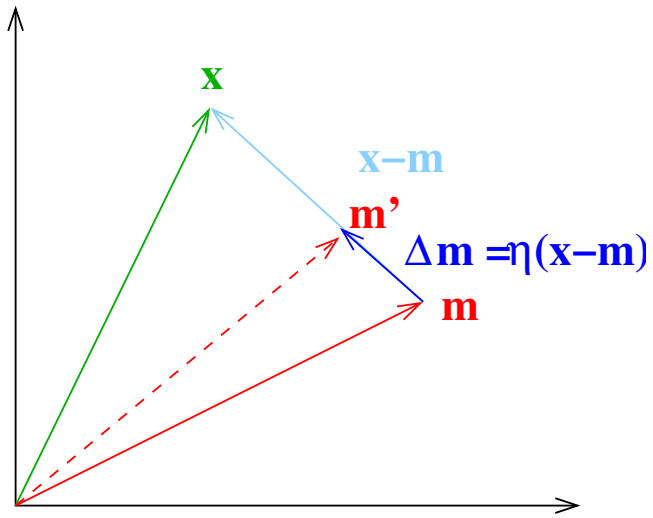


- $\mathcal{X} = \{\mathbf{x}^t\}_t$: set of samples (green).
- $\mathbf{m}_i, i = 1, 2, ..., k$: cluster centers (red).
- b_i^t: if $\mathbf{m}_i$ is closest to $\mathbf{x}^t$, 1.
- Note: t = index for input, i = index for cluster center.

Competetive Learning: k-Means

- Batch: update cluster centers according to simple "mean" at each moment.
- Online: Use stochastic gradient descent.
  - Note: $\mathbf{m}_i$ is a vector, having scalar components m_{ij} (see next page).
- Both are iteratively done until convergence is achieved.

Competetive Learning



- Updating the center.
- $\Delta \mathbf{m} = \eta(\mathbf{x} - \mathbf{w})$
- $\mathbf{m} = \mathbf{m} + \eta(\mathbf{x} - \mathbf{w})$
- “Move center closer to the current input”

Competitive Learning

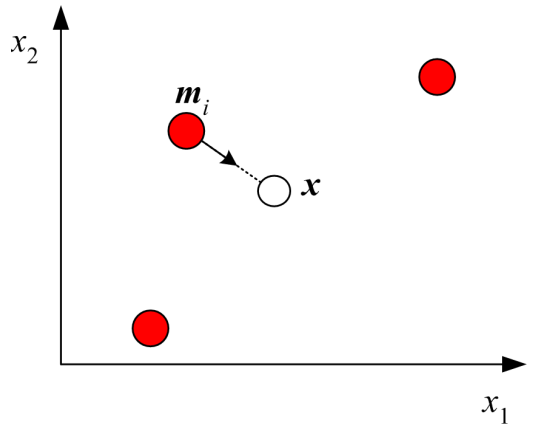
$$E\left(\{\mathbf{m}_i\}_{i=1}^k | \mathcal{X}\right) = \sum_t \sum_i b_i^t \left\|\mathbf{x}^t - \mathbf{m}_i\right\|$$

$$b_i^t = \begin{cases} 1 & \text{if } \left\|\mathbf{x}^t - \mathbf{m}_i\right\| = \min_l \left\|\mathbf{x}^t - \mathbf{m}_l\right\| \\ 0 & \text{otherwise} \end{cases}$$

Batch k-means: $\mathbf{m}_i = \dfrac{\sum_t b_i^t \mathbf{x}^t}{\sum_t b_i^t}$

Online k-means:

$$\Delta m_{ij} = -\eta \frac{\partial E^t}{\partial m_{ij}} = \eta b_i^t \left(x_j^t - m_{ij}\right)$$



Replacing b_i^t, etc.

- We can use *lateral inhibition* to implement b_i^t in a more biologically plausible manner (see figure in next slide). Needs iteration until vlaues settle.
- We can also use dot product instead of Euclidean distance as a distance measure.
- Hebbian learning is usually used for biologically plausible models.

Initialize $\boldsymbol{m}_i, i = 1, \ldots, k$, for example, to k random $\boldsymbol{x}^t$

Repeat

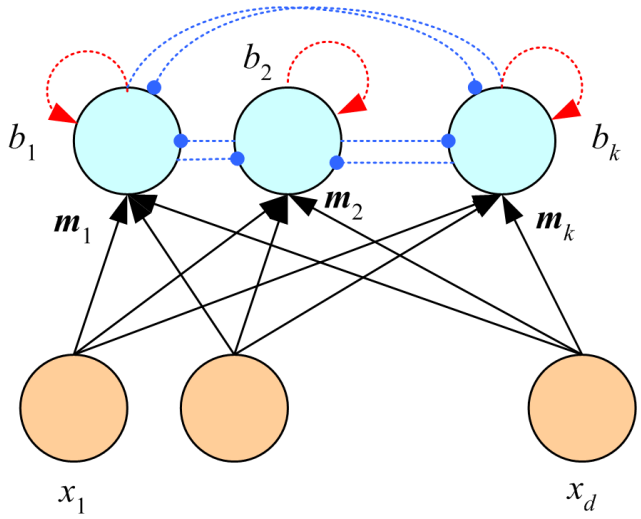
  For all $\boldsymbol{x}^t \in \mathcal{X}$ in random order

    $i \leftarrow \arg\min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\|$

    $\boldsymbol{m}_i \leftarrow \boldsymbol{m}_i + \eta(\boldsymbol{x}^t - \boldsymbol{m}_i)$

Until $\boldsymbol{m}_i$ converge

Winner-take-all network

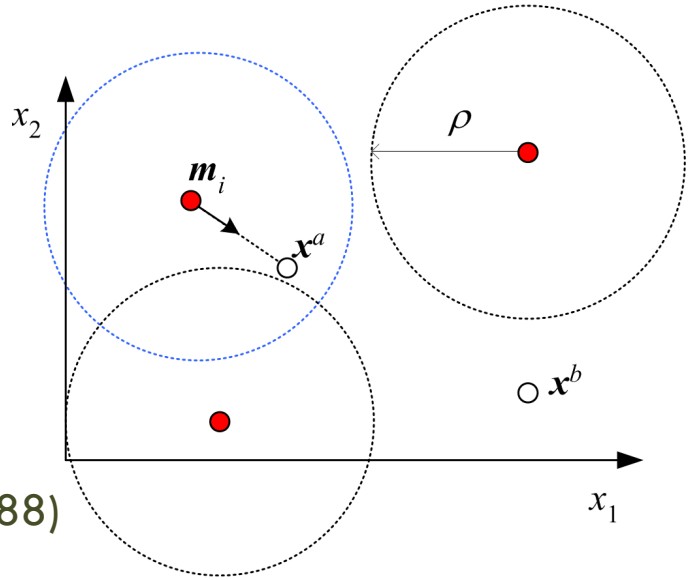


Adaptive Resonance Theory

- Incremental; add a new cluster if not covered; defined by vigilance, ρ

$$b_i^t = \left\|\mathbf{x}^t - \mathbf{m}_i\right\| = \min_{l=1}^{k} \left\|\mathbf{x}^t - \mathbf{m}_l\right\|$$

$$\begin{cases} \mathbf{m}_{k+1} \leftarrow \mathbf{x}^t & \text{if } b_i > \rho \\ \Delta\mathbf{m}_i = \eta\left(\mathbf{x}^t - \mathbf{m}_i\right) & \text{otherwise} \end{cases}$$



(Carpenter and Grossberg, 1988)

SOM Overview

SOM is based on three principles:

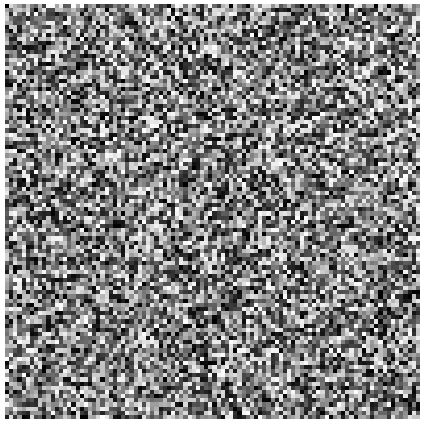
- **Competition**: each neuron calculates a discriminant function. The neuron with the highest value is declared the winner.
- **Cooperation**: Neurons near-by the winner on the lattice get a chance to adapt.
- **Adaptation**: The winner and its neighbors increase their discriminant function value relative to the current input. Subsequent presentation of the current input should result in enhanced function value.

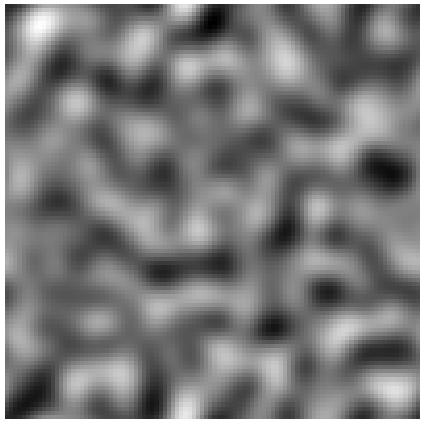
Redundancy in the input is needed!

Redundancy, etc.

- Unsupervised learning such as SOM require redundancy in the data.
- The following are intimately related:
  - Redundancy
  - Structure (or organization)
  - Information content relative to channel capacity

Redundancy, etc. (cont'd)

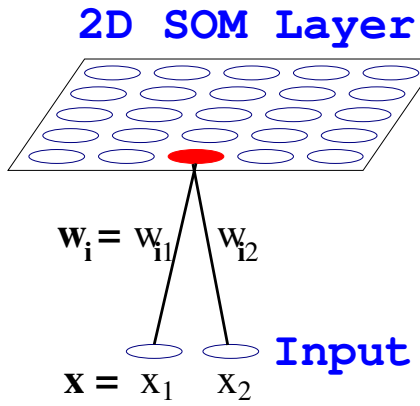




	Left	Right
Structure	No	Yes
Redundancy	No	Yes
Info $<$ Capacity	No	Yes

Consider each pixel as one random variable.

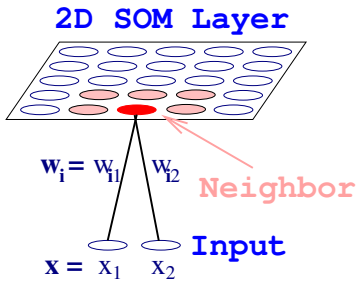
Self-Organizing Map (SOM)



Kohonen (1982)

- 1-D, 2-D, or 3-D layout of units.
- One weight vector for each unit.
- Unsupervised learning (no target output).

SOM Algorithm



1. Randomly initialize weight vectors $\mathbf{w}_i$
2. Randomly sample input vector $\mathbf{x}$
3. Find Best Matching Unit (BMU):

$$i(\mathbf{x}) = \text{argmin}_j \|\mathbf{x} - \mathbf{w}_j\|$$

4. Update weight vectors:

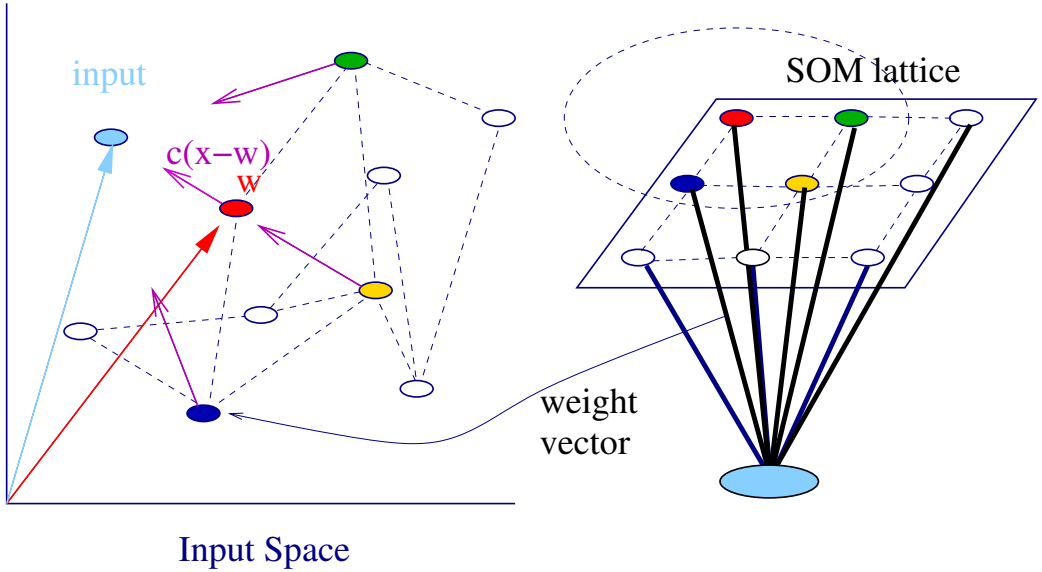
$$\mathbf{w}_j \leftarrow \mathbf{w}_j + \eta h(j, i(\mathbf{x}))(\mathbf{x} - \mathbf{w}_j)$$

η : learning rate

$h(j, i(\mathbf{x}))$: neighborhood function of BMU.

5. Repeat steps 2 – 4.

SOM Learning

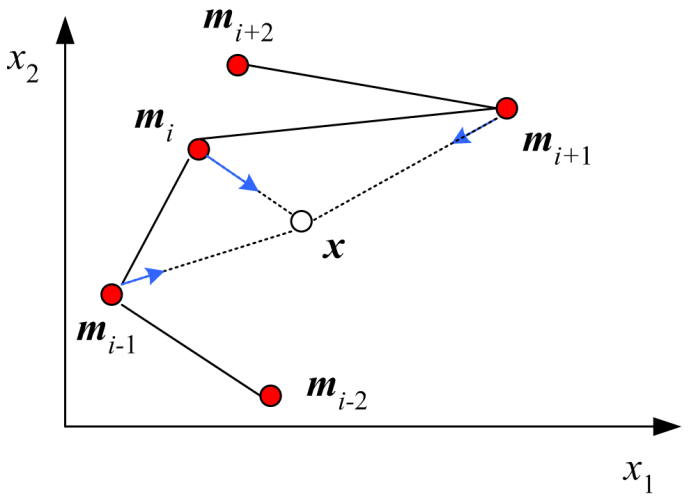


- Weight vectors can be plotted in the input space.
- Weight vectors move, not according to their proximity to the input in the input space, but according to their proximity in the lattice.

Self-Organizing Maps

- Units have a neighborhood defined; $\boldsymbol{m}_i$ is "between" $\boldsymbol{m}_{i-1}$ and $\boldsymbol{m}_{i+1}$, and are all updated together
- One-dim map:

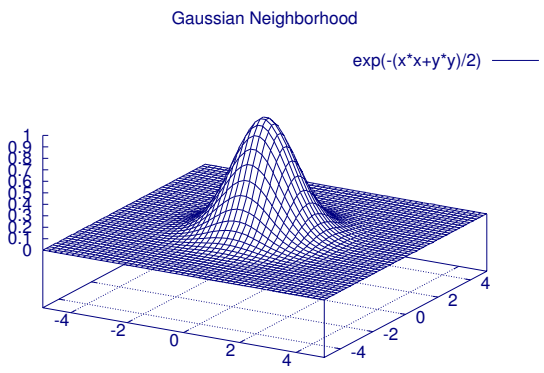
(Kohonen, 1990)



$$\Delta \mathbf{m}_l = \eta e(l,i)\left(\mathbf{x}^t - \mathbf{m}_l\right)$$

$$e(l,i) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left[-\frac{(l-i)^2}{2\sigma^2}\right]$$

Typical Neighborhood Functions

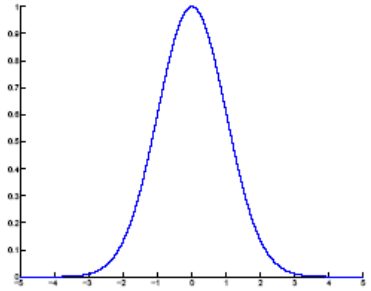


- Gaussian: $h(j, i(\mathbf{x})) = \exp(-\|\mathbf{r}_j - \mathbf{r}_{i(\mathbf{x})}\|^2/2\sigma^2)$
- Flat: $h(j, i(\mathbf{x})) = 1$ if $\|\mathbf{r}_j - \mathbf{r}_{i(\mathbf{x})}\| \leq \sigma,$ and 0 otherwise.
- σ is called the **neighborhood radius**.
- $\mathbf{r}_j$ is the location of unit j on the lattice.

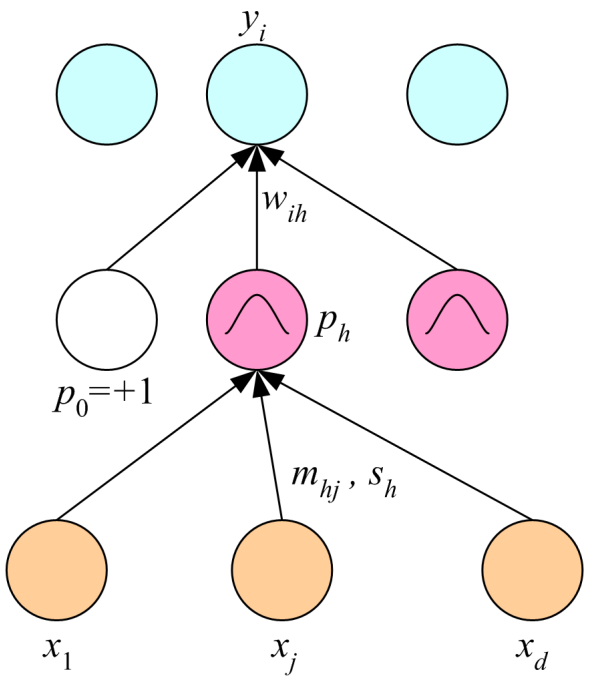
Radial-Basis Functions

- Locally-tuned units:

$$p_h^t = \exp\left[-\frac{\left\|\mathbf{x}^t - \mathbf{m}_h\right\|^2}{2s_h^2}\right]$$



$$y^t = \sum_{h=1}^{H} w_h p_h^t + w_0$$



RBF

- Input x to p: cluster centers $\mathbf{m}$ and radius (variance) s are estimated.
- p to output weights $\mathbf{w}$ can be calculated in one shot using pseudo inverse (output units are usually linear units). n RBF activation values (each row in P is the RBF activation values generated from each input vector), H RBF units, m output units.

$$\begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1H} \\ p_{21} & p_{22} & \cdots & p_{2H} \\ \vdots & \vdots & \vdots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nH} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_H \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix},$$

$$\mathrm{P}\mathbf{w} = \mathbf{y}$$

$$\mathbf{w} = \mathrm{P}^{-1}\mathbf{y}, \text{ if } n = H$$

$$\mathbf{w} = \left(\mathrm{P}^T\mathrm{P}\right)^{-1}\mathrm{P}^T\mathbf{y}, \text{ if } n > H$$

- Note: $(P^TP)^{-1}P^TP = (P^{-1}(P^T)^{-1})P^TP = P^{-1}(P^T)^{-1}P^TP = P^{-1}P = I$
- Other iterative methods also exist (see next few slides).

Training RBF

- Hybrid learning:
  - First layer centers and spreads:

    Unsupervised *k*-means
  - Second layer weights:

    Supervised gradient-descent
- Fully supervised

(Broomhead and Lowe, 1988; Moody and Darken, 1989)

Regression

$$E\left(\{\mathbf{m}_h, s_h, w_{ih}\}_{i,h} \mid \mathcal{X}\right) = \frac{1}{2}\sum_t \sum_i \left(r_i^t - y_i^t\right)^2$$

$$y_i^t = \sum_{h=1}^{H} w_{ih} p_h^t + w_{i0}$$

$$\Delta w_{ih} = \eta \sum_t \left(r_i^t - y_i^t\right) p_h^t$$

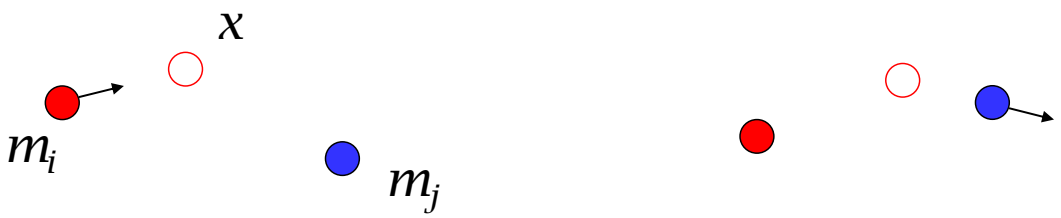
$$\Delta m_{hj} = \eta \sum_t \left[\sum_i \left(r_i^t - y_i^t\right) w_{ih}\right] p_h^t \frac{\left(x_j^t - m_{hj}\right)}{s_h^2}$$

$$\Delta s_h = \eta \sum_t \left[\sum_i \left(r_i^t - y_i^t\right) w_{ih}\right] p_h^t \frac{\left\|\mathbf{x}^t - \mathbf{m}_h\right\|^2}{s_h^3}$$

Learning Vector Quantization

- *H* units per class prelabeled (Kohonen, 1990)
- Given $\mathbf{x}$, $\mathbf{m}_i$ is the closest:

$$\begin{cases} \Delta\mathbf{m}_i = \eta\left(\mathbf{x}^t - \mathbf{m}_i\right) & \text{if label}(\mathbf{x}^t) = \text{label}(\mathbf{m}_i) \\ \Delta\mathbf{m}_i = -\eta\left(\mathbf{x}^t - \mathbf{m}_i\right) & \text{otherwise} \end{cases}$$



References