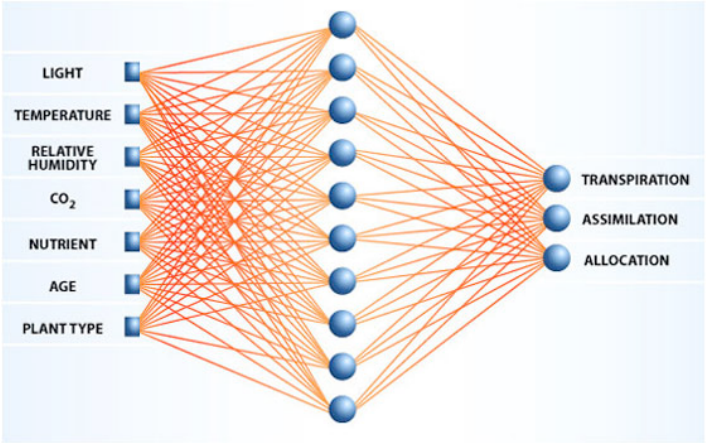
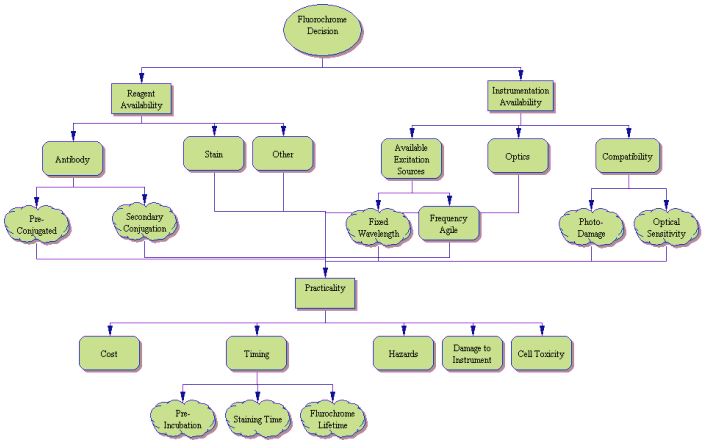
Apprendimento Supervisionato

Tecniche:

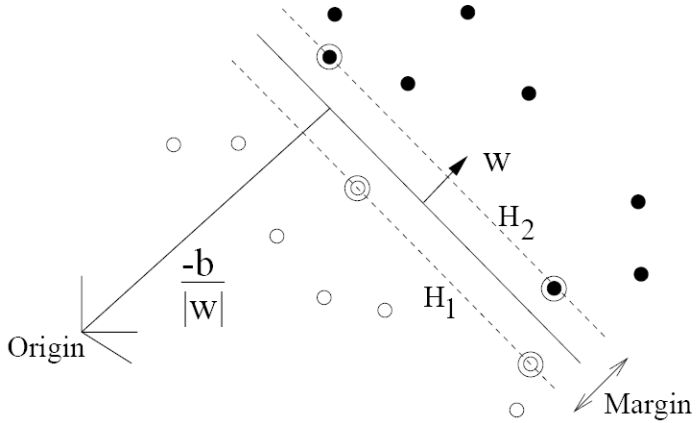
Neural Networks



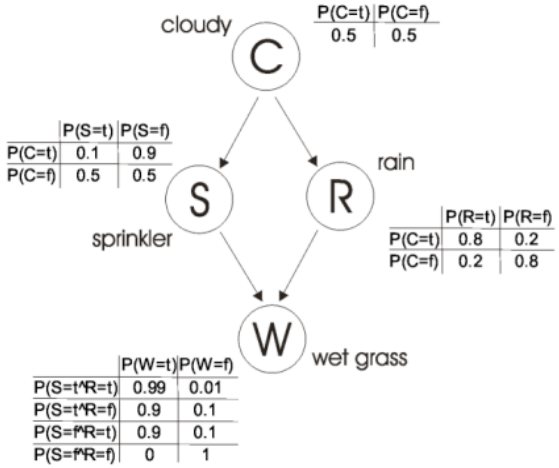
Decision Trees



Support Vector Machines



Bayesian Networks



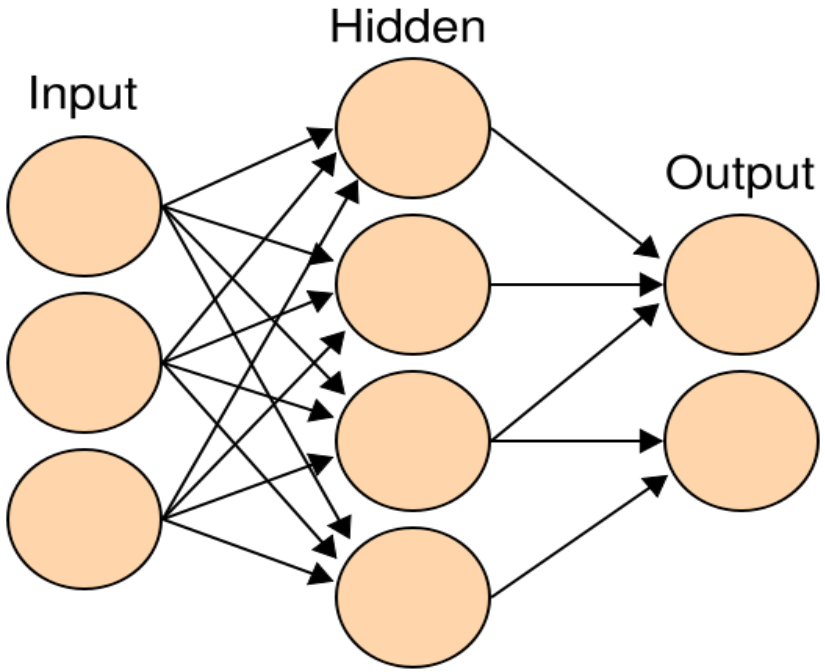
Reti Neurali Artificiali

Modello Computazionale basato sulle reti neurali biologiche.

Rete di neuroni artificiali interconnessi.

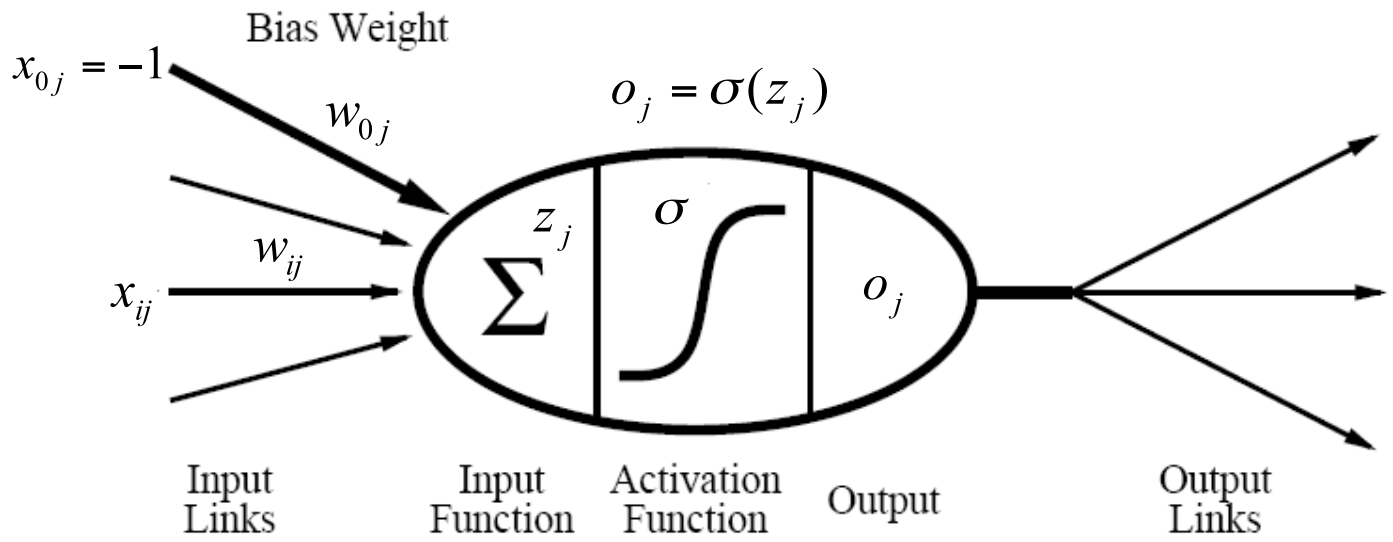
Classificazione / Regressione.

Modello adattivo:
Adatta il suo stato interno alle coppie input-output di training



Origini

Modello matematico di neurone:

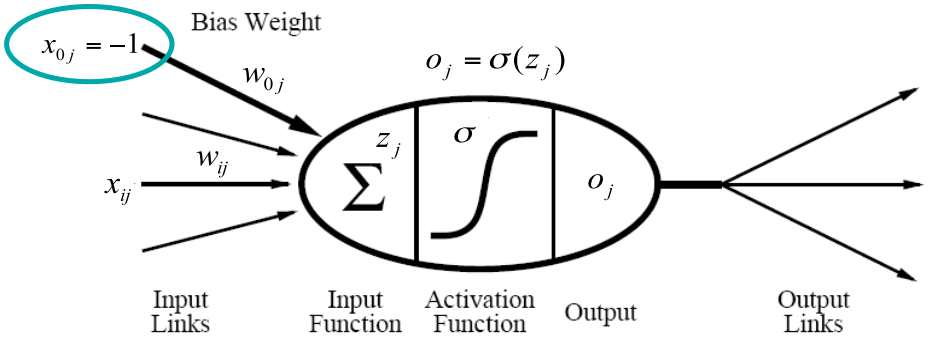


McCulloch, W. S. and Pitts, W. (**1943**). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*, 115–137.

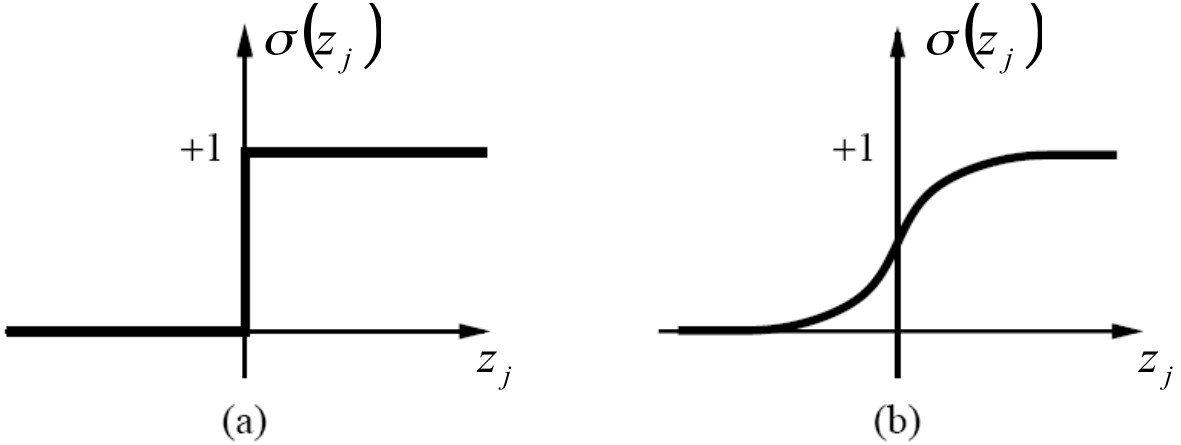
Neurone

Per il j-mo neurone:

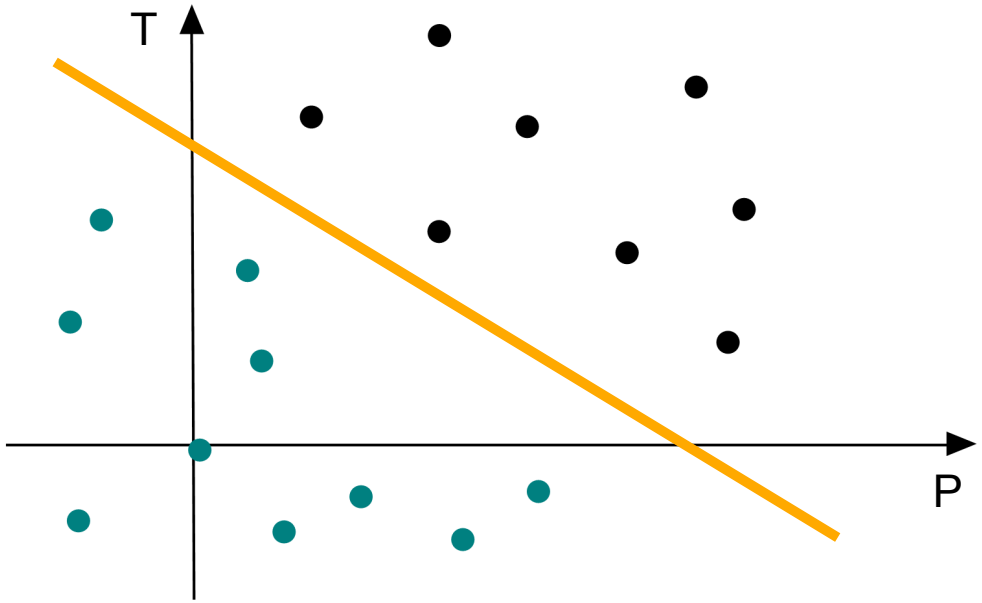
- Input: $\mathbf{x}_j = \{-1, x_{1j}, \ldots, x_{nj}\}$
- Pesi: $\mathbf{w}_j = \{w_{0j}, w_{1j}, \ldots, w_{nj}\}$
- Output: o_j



$$z_j = \sum_{i=0}^{n} w_{ij} x_{ij} = \mathbf{w}_j \cdot \mathbf{x}_j \qquad o_j = \sigma(z_j) = \sigma\left(\sum_{i=0}^{n} w_{ij} x_{ij}\right)$$



Separatore Lineare



$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = w_0$$

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n - 1 \cdot w_0 = 0$$

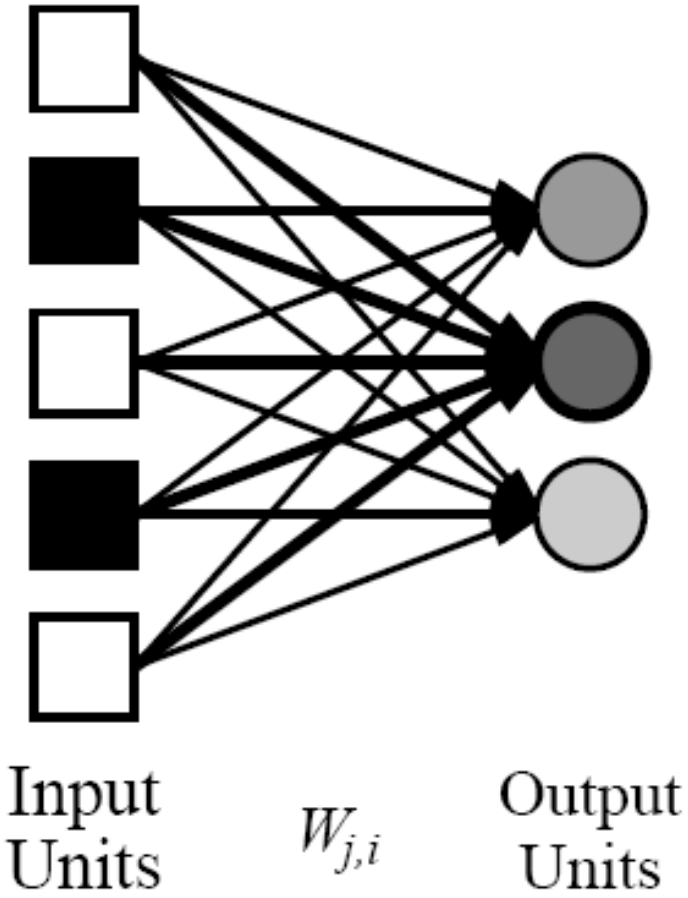
$$\mathbf{w} \cdot \mathbf{x} = 0$$

$$\sigma(\mathbf{w} \cdot \mathbf{x}) = \begin{cases} 1 & \mathbf{w} \cdot \mathbf{x} \geq 0 \\ 0 & \mathbf{w} \cdot \mathbf{x} < 0 \end{cases}$$

Percettrone

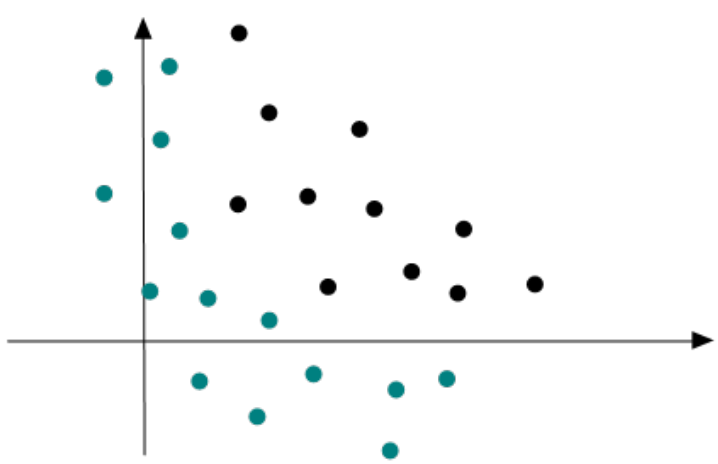
- Rete neurale a singolo strato
- Un neurone in uscita per ogni classe
- Consente di classificare datasets linearmente separabili

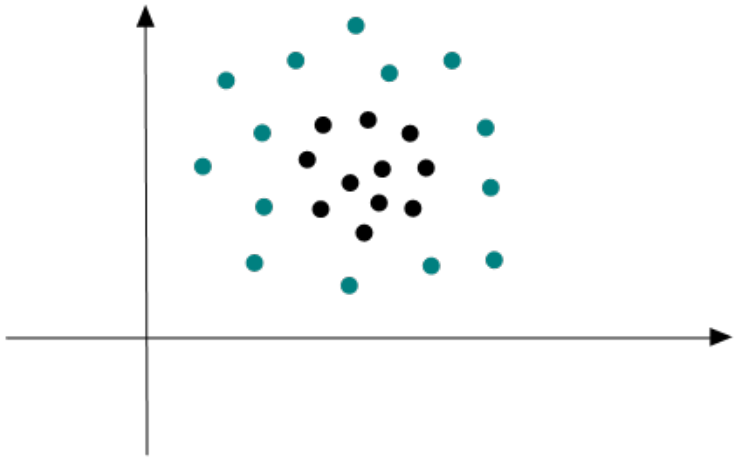
E se ho dati non linearmente separabili?

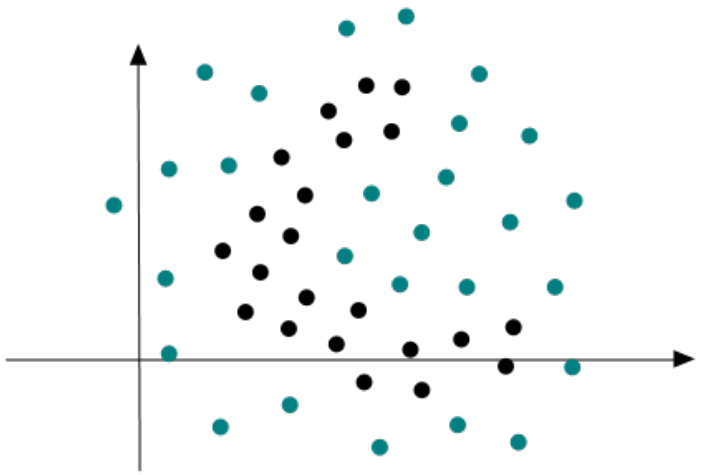


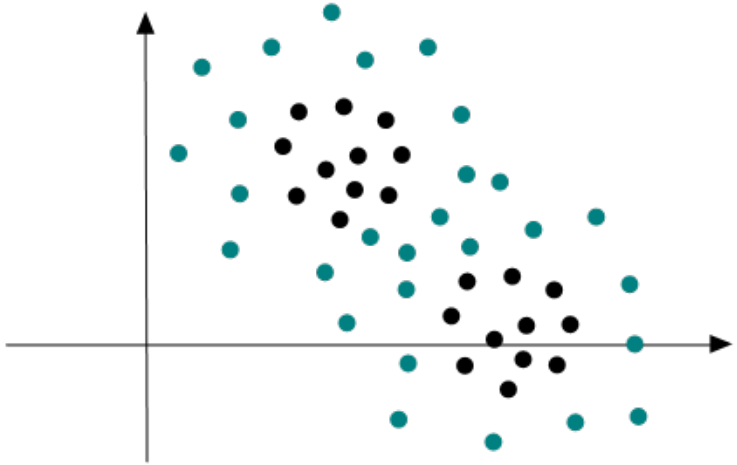
Rosenblatt, F. (**1957**). The perceptron: A perceiving and recognizing automaton. *Report 85-460-1, Project PARA, Cornell Aeronautical Laboratory, Ithaca, NewYork.*

Dati non linearmente separabili

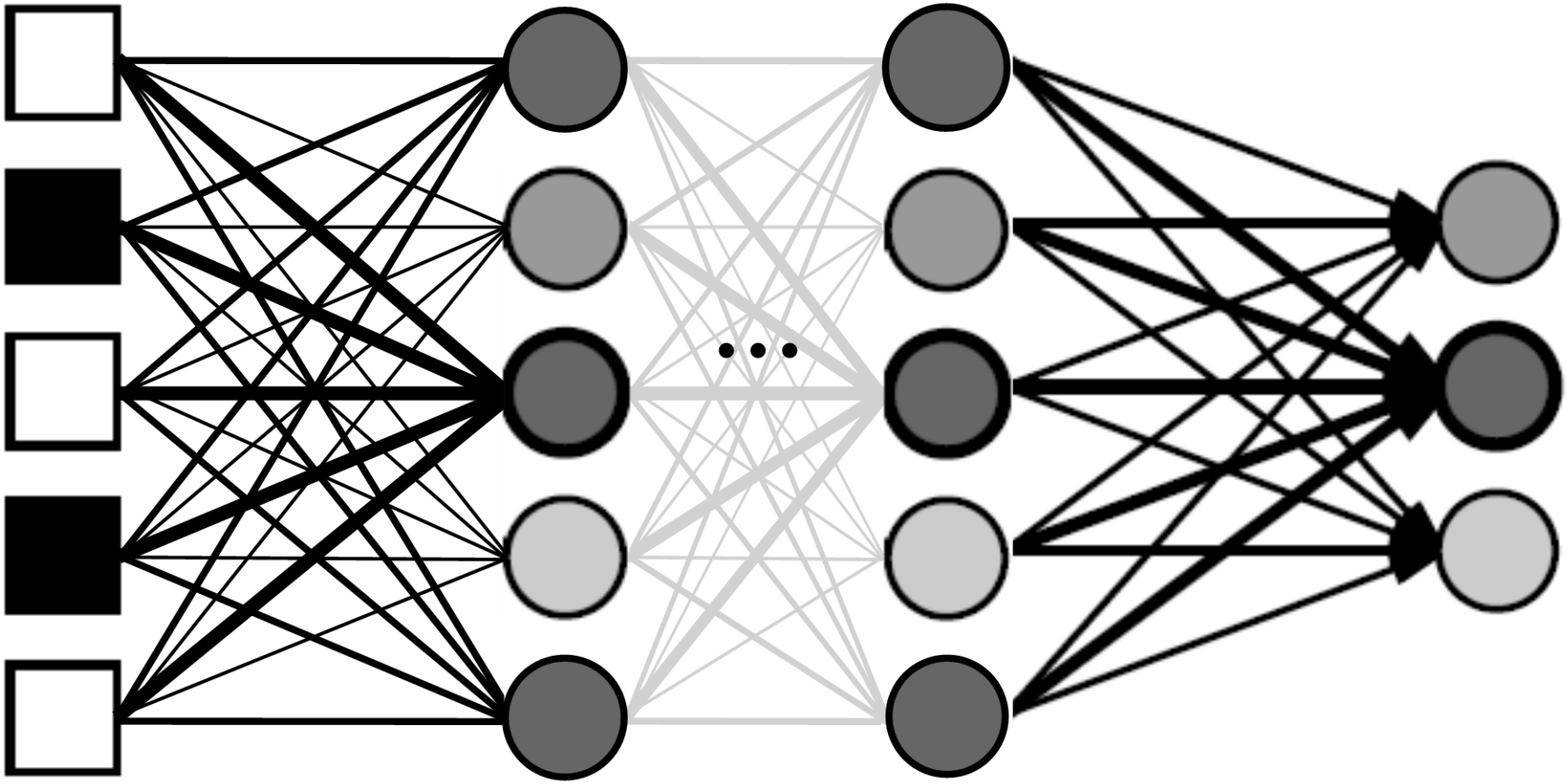








Reti Neurali Multistrato



Backpropagation

Algoritmo di allenamento di una rete a partire dalle coppie (***x***,y) (training set).

Sottopone più volte il training set alla rete, aggiustando i pesi per minimizzare l'errore quadratico.

Algoritmo gradient descent, efficiente ma può arenarsi in un ottimo locale.

L'allenamento è, in generale, un problema NP-Completo.

```
Initialize weights at random
repeat
    for each example in the training set
        compute example's output
        compute quadratic error
        for i = levels_# down to 1
            compute update for weights
            at level i
        end
        update all weights
    end
until (all examples correctly classified
or max iterations reached)
```


Werbos (**1974**). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. *Ph.D. Thesis, Harvard University.*

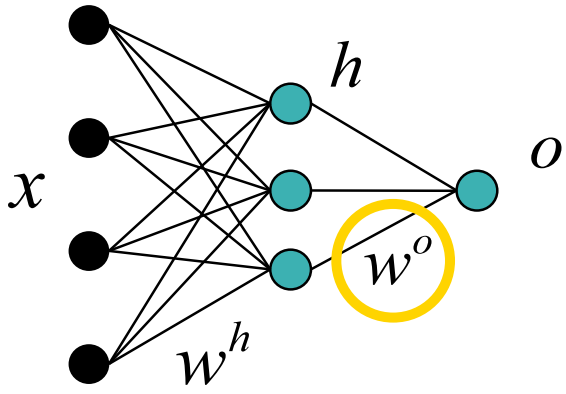
Rumelhart, Hintont, Williams (**1986**). Learning representations by back-propagating errors. *Nature*

Backpropagation

Definizioni:

$$z_j^h = \sum_{i=0}^{n} w_{ij}^h x_i \qquad h_j = \sigma(z_j^h)$$

$$z^o = \sum_{j=0}^{m} w_j^o h_j \qquad o = \sigma(z^o)$$



Funzione di attivazione:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

$\boldsymbol{x} \in \mathbb{R}^{n,1} \quad \boldsymbol{w^h} \in \mathbb{R}^{n,m}$

$\boldsymbol{h} \in \mathbb{R}^{m,1} \quad \boldsymbol{w^o} \in \mathbb{R}^{1,m}$

Errore:

$$E = \frac{1}{2}(y-o)^2 \qquad w_i = w_i - \alpha\frac{\partial E}{\partial w_i} = w_i + \Delta w_i$$

$$\frac{\partial E}{\partial w_j^o} = \frac{\partial E}{\partial o} \cdot \frac{\partial o}{\partial z^o} \cdot \frac{\partial z^o}{\partial w_j}$$

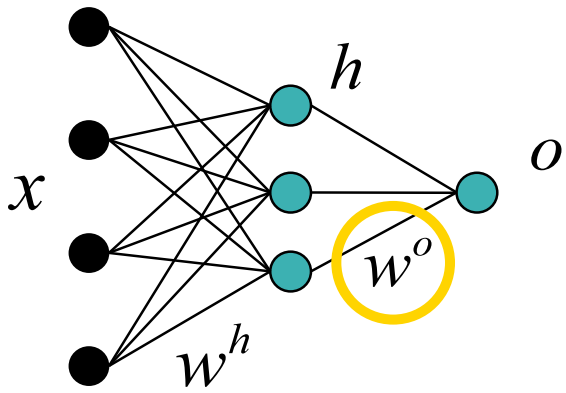
Backpropagation

$$\frac{\partial E}{\partial w_j^o} = \frac{\partial E}{\partial o} \cdot \frac{\partial o}{\partial z^o} \cdot \frac{\partial z^o}{\partial w_j}$$

$$z^o = \sum_{j=0}^{m} w_j^o h_j$$

$$o = \sigma(z^o)$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$



$$\frac{\partial E}{\partial o} = \frac{\partial}{\partial o}\left[\frac{1}{2}(y - o)^2\right] = -(y - o)$$

$$\frac{\partial o}{\partial z^o} = o \cdot (1 - o)$$

$$\frac{\partial z^o}{\partial w_j^o} = h_j$$

$$\frac{\partial E}{\partial w_j^o} = -(y - o) \cdot o \cdot (1 - o) \cdot h_j = -\boldsymbol{\delta^o} h_j$$

Aggiornamento del peso:

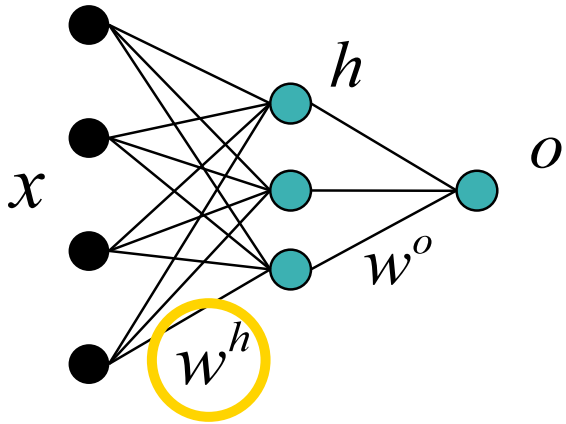
$$\Delta w_j^o = \alpha \delta^o h_j$$

Backpropagation

$$\frac{\partial E}{\partial w_{ij}^h} = \frac{\partial E}{\partial o} \cdot \frac{\partial o}{\partial z^o} \cdot \frac{\partial z^o}{\partial h_j} \cdot \frac{\partial h_j}{\partial z_j^h} \cdot \frac{\partial z_j^h}{\partial w_{ij}^h}$$

$$z_j^h = \sum_{i=0}^{n} w_{ij}^h x_i$$

$$h_j = \sigma(z_j^h)$$



$$\frac{\partial E}{\partial o} \cdot \frac{\partial o}{\partial z^o} = -\delta^o$$

$$\frac{\partial z^o}{\partial h_j} = w_j^o$$

$$\frac{\partial h_j}{\partial z_j^h} = h_j \cdot (1 - h_j)$$

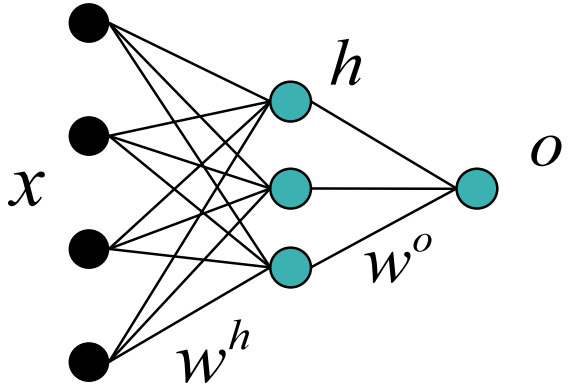
$$\frac{\partial z_j^h}{\partial w_{ij}^h} = x_i$$

$$\frac{\partial E}{\partial w_{ij}^h} = -\delta^o \cdot w_j^o \cdot h_j \cdot (1 - h_j) \cdot x_i = -\delta_j^h x_i$$

Aggiornamento del peso:

$$\Delta w_{ij}^h = \alpha \delta_j^h x_i$$

Backpropagation



Aggiornamento dei pesi:

$$\Delta w_j^o = \alpha \delta^o h_j$$

$$\Delta w_{ij}^h = \alpha \delta_j^h x_i$$

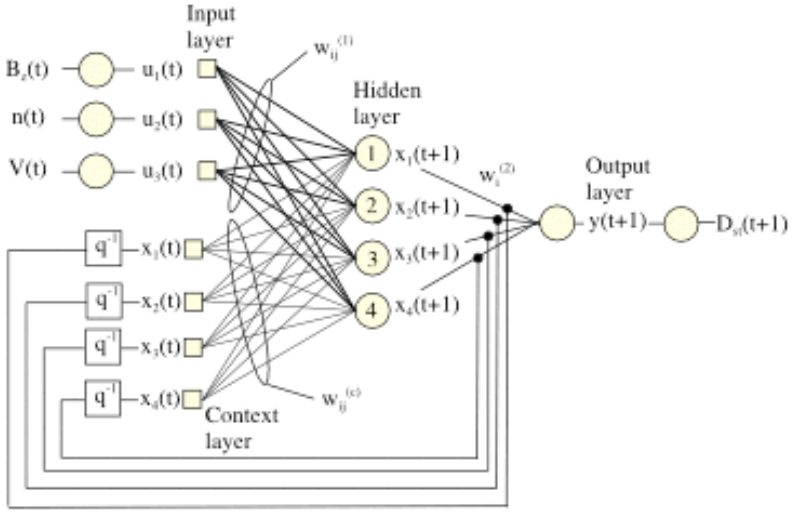
con

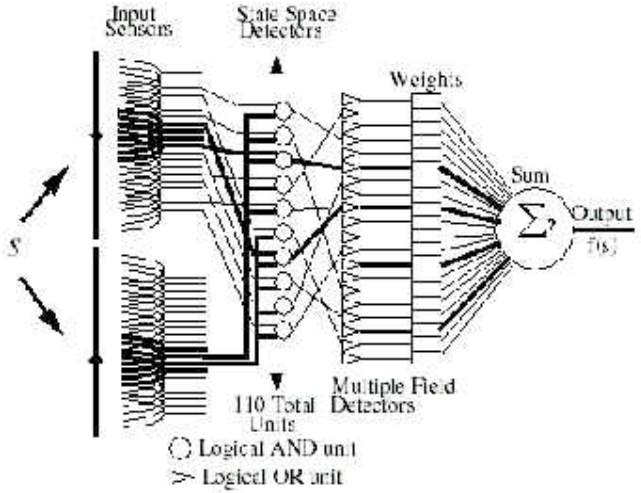
$$\delta^o = (y - o) \cdot o \cdot (1 - o)$$

$$\delta_j^h = \delta^o \cdot w_j^o \cdot h_j \cdot (1 - h_j)$$

Altre tipologie di Reti Neurali

Recurrent Neural Networks





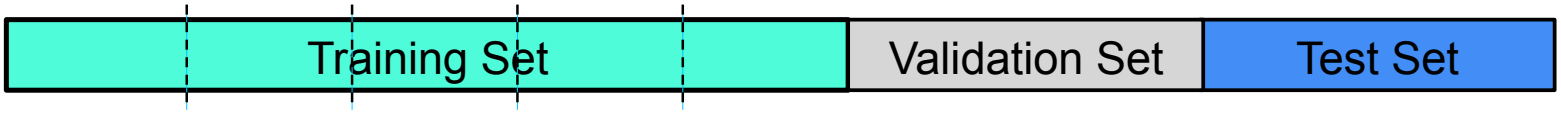
Associative Neural Networks

Stochastic Neural Networks

Spiking Neural Networks

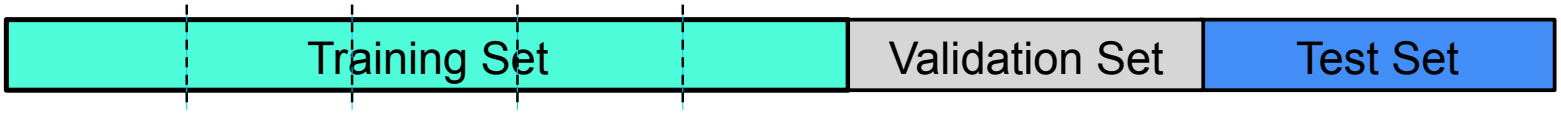
…

Procedura di apprendimento – Reti Neurali



- 3 sottoinsiemi: Training, Validation e Test Set
- # nodi in ingresso = # features
- # nodi in uscita = # di classi
- # hidden layer e # nodi per livello: k-fold cross validation sul training set.
- Alleno la struttura scelta con tutto il training set, limitando l'overfitting col validation set.
- Valuto l'accuratezza finale sul test set.

Buone abitudini e regolette euristiche



- 1 hidden layer è sufficiente per la stragrande maggioranza dei problemi (e l'allenamento è più rapido)
- Se devo scegliere il numero di nodi interni, parto con pochi e cresco (esponenzialmente) finchè vedo un miglioramento

  per es. 5 10 20 50 100 …