

## AGENTE :

- strettamente legato al concetto di ambiente
- e' un modo di vedere il software
- e' in grado di percepire l'ambiente
- tramite percezione e altre conoscenze effettua l'operazione di deliberazione (= decidere cosa fare / che risposta restituire) e infine viene compiuta una azione che avra' un effetto sull'ambiente

Parleremo solo di sistemi mono-agente.

- ci aspettiamo che l'agente abbia un comportamento razionale (weak-IA)  
cioe' orientato al conseguimento di un compito
- programmato ad alto livello

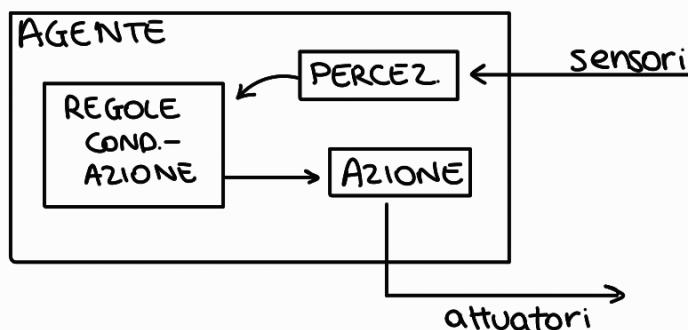
⚠️ razionale → ottimo perché le scelte degli agenti sono fortemente determinate dalla loro conoscenza

razionale → ottimizzazione del risultato atteso  
(non del risultato reale)

## Tipologie di agente

### AGENTE REATTIVO SEMPLICE

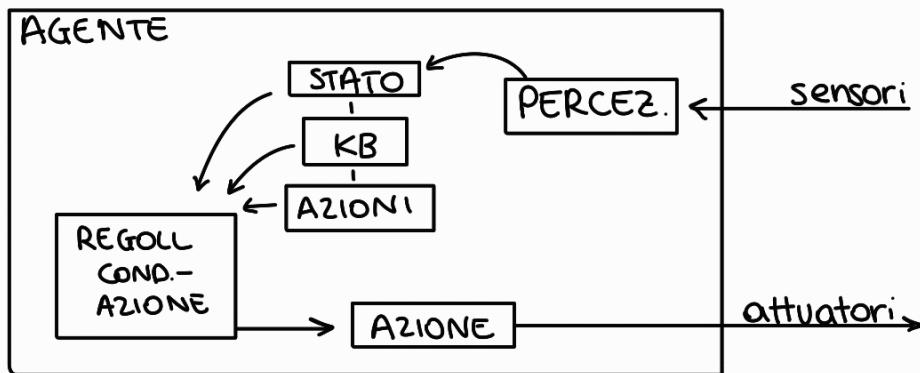
- 1) grazie a dei sensori percepisce come e' il mondo
- 2) passa la lettura a un insieme di regole condizione-azione
- 3) si sceglie l'azione che verrà applicata tramite attuatori



Viene usato quando si ha una percezione completa, ovvero quando l'ambiente e' completamente osservabile

## AGENTE REATTIVO BASATO SU MODELLO

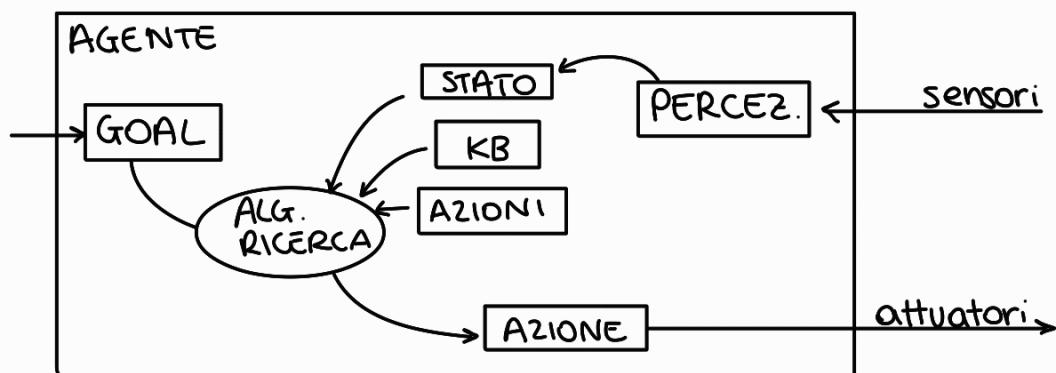
Oltre ai sensori, per percepire l'ambiente, ha un'informazione di stato e una KB sul mondo che gli suggerisce come l'ambiente evolve, e ha conoscenza delle proprie azioni.



Limite: svolge un compito unico

## AGENTE GUIDATA DAGLI OBIETTIVI

Invece di avere delle regole di condizione-azione fisse, viene programmato per obiettivi



## AGENTE GUIDATA DALL'UTILITÀ

Utilità: funzione che dice "se siamo felici"  
↳ da massimizzare

## AGENTE CHE APPRENDE

Modifica la sua KB

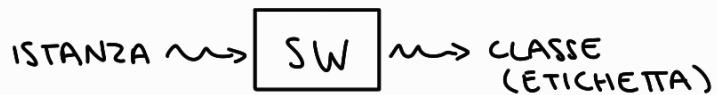
## APPRENDIMENTO AUTOMATICO

- disciplina molto vasta , contanti approcci

- simbolici
- subsimbolici
- reti neurali
- classificazione
- alberi di decisione
- random forest
- regole
- algoritmi genetici
- apprendimenti bayesiani
- apprendimento per rinforzo
- ...

## CLASSIFICAZIONE

Abbiamo bisogno di interpretare le istanze del mondo secondo determinate categorie



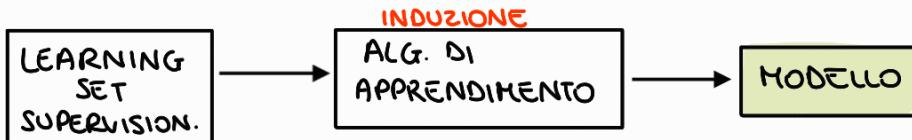
- Mostriamo al programma degli esempi (istanze) con le relative etichette  $\rightsquigarrow$
- Il programma deve generalizzare , trovare degli schemi  
 $\rightarrow$  PROCESSO DI INDUZIONE

Classificazione: problema di apprendimento supervisionato

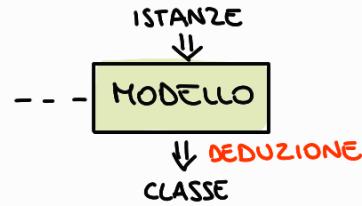


**learning set**

ISTANZA	nome	temperatura	Copertura pelle	viviparo	Creatura acquatica	Creatura volatile	zampe	letargo	CLASSE
	uomo	Sangue caldo	peluria	sì	no	no	sì	no	mammifero
	pitone	Sangue freddo	squame	no	no	no	no	sì	rettile
	salmone	Sangue freddo	squame	no	sì	no	no	no	pesce
	balena	Sangue caldo	peluria(?)	sì	sì	no	no	no	mammifero
	rana	Sangue freddo	nessuna	no	semi	no	sì	sì	anfibio
	pinguino	Sangue caldo	piumaggio	no	semi	no	sì	no	uccello
	piccione	Sangue caldo	piumaggio	no	no	sì	sì	no	uccello



Modello: rappresentazione generalizzata a cui vogliamo dare in input delle nuove istanze da classificare



Valutazione del modello : gli si sottopone un ulteriore insieme di esempi , detto **test set**, che non comprende però le classi di

appartenenza.  
buon modello

se fornisce risposte coerenti con l'insieme delle risposte attese

si utilizza una **matrice di confusione** → matrice quadrata NxN

classi fornite dal sistema

con N= # classi

classi attese		classi fornite dal sistema	
$n_{11}$	$n_{12}$	$n_{13}$	...
$n_{21}$	$n_{22}$		
		$n_{33}$	
:			..

$n_{ij}$  = numero delle istanze di classe i riconosciute come appartenenti alla classe j

quando  $i \neq j \rightarrow$  classe sbagliata

Un modello e' ottimo quando abbiamo tutti i valori  $\neq 0$  concentrati sulla **diagonale principale**

Sulla matrice di confusione vengono calcolati due indici :

$$\cdot \text{ACCURATEZZA} = \frac{\sum_{i=1}^N n_{ii}}{T} \quad T : \# \text{ istanze del test set}$$

↪ perfetto se = 1

$$\cdot \text{ERROR RATE} = \frac{\sum_{i \neq j}^N n_{ij}}{T}$$

A volte alla matrice di confusione si associa una **matrice dei costi** (stessa cardinalità della matr. di confusione) contenente dei pesi che indicano la gravità dell'errore

- Rote learning** = apprendimento meccanico = imparare a memoria
- non generalizza, si limita a memorizzare dei casi dal learning set.
  - le istanze memorizzate vengono usate per fare confronti con l'istanza della query
    - se la trova : restituisce la classe corrispondente
    - altrimenti : ne cerca una simile → possibile ambiguità

Algoritmi di apprendimento diversi producono modelli di tipo oliverso.

### ALBERO DI DECISIONE

- Costruisce un modello per induzione
- Modello ≡ albero
  - nodi interni ≡ test
  - foglie ≡ classi

Data un'istanza, si percorre l'albero a seconda di come risponde ai vari test che incontra. Quando si raggiunge una foglia, si restituisce la classe corrispondente.

Ogni generico nodo interno è associato ad un attributo descrittivo usato nel learning set

L'albero può essere tradotto in un insieme di regole IF-THEN

# Algoritmo di Hunt

L'albero viene costruito procedendo *ricorsivamente* e *suddividendo il learning set in sottoinsiemi via via più "puri"*. Il generico passo di suddivisione di un nodo dell'albero esegue questi passi:

**Dati:**

$D_t$  = sottoinsieme del learning set associato al nodo  $t$

$y = \{y_1, y_2, \dots, y_c\}$  = insieme delle etichette che identificano le classi

**passo 1:** se tutte le istanze in  $D_t$  appartengono alla stessa classe  $y_t$  allora il nodo è una foglia etichettata dalla classe  $y_t$  delle sue istanze

**passo 2:** si sceglie un attributo fra quelli che descrivono le istanze, *si produce un nodo figlio per ogni possibile valore dell'attributo* (il range è dato dal learning set?).  $= \text{SPLIT}$

A ciascun nodo figlio si associa uno specifico valore e si associano ad esso anche quelle istanze, già associate al padre, per le quali l'attributo assume il valore corrispondente al nodo

# Algoritmo di Hunt

combinazioni non previste

**nota 1:** se una certa combinazione di valori non è rappresentata da nessuna istanza, questa sarà associata alla **classe di default** (se esiste)

oppure devo arricchire il L.S.

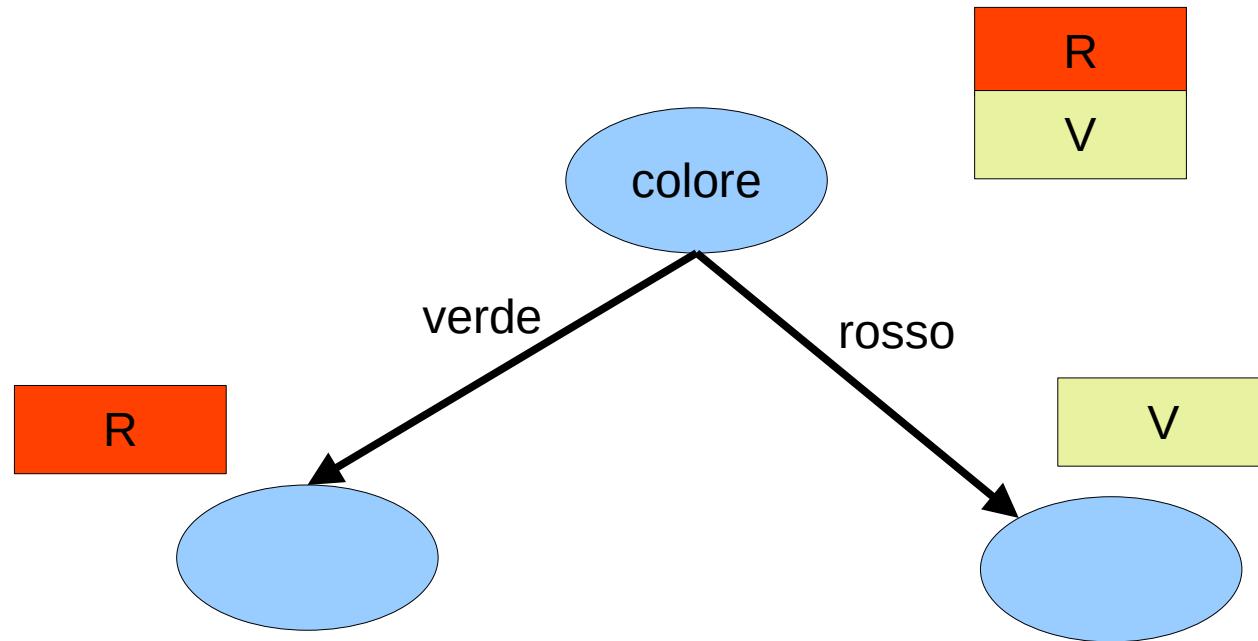
**nota 2:** se **tutte le istanze** associate a un nodo sono **identiche** come tuple **ma corrispondono a classi differenti** (non-determinismo), il nodo non può essere scisso.

Diventa una foglia che ha associata la classe più rappresentata

**nota 3:** quando si termina la costruzione dell'albero?

**nota 4:** come si sceglie l'attributo di split?

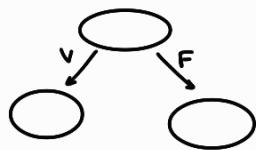
# Split = suddividere il learning set



## TIPOLOGIE DI SPLIT:

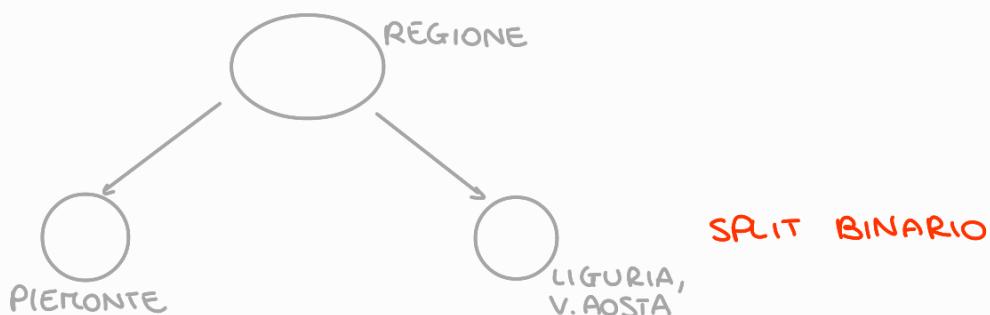
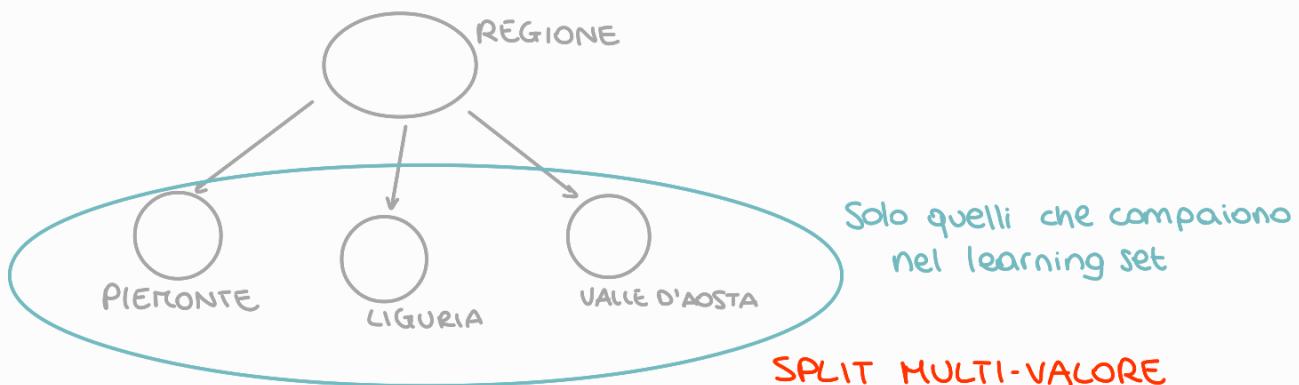
- Nominali
- Binari
- Ordinali
- Continui

### Attributi binari



solo due nodi figli

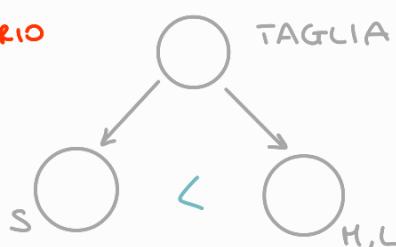
### Attributi nominali : valori = etichette



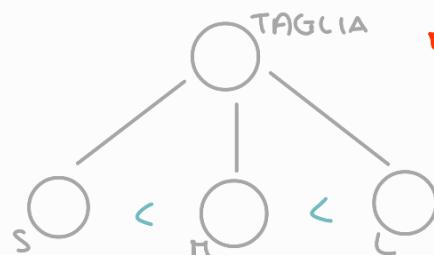
### Attributi ordinali : vale una relazione d'ordine

TAGLIA: S M L

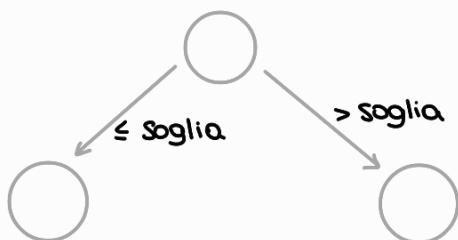
BINARIO



MULTIVALEORE



### Attributi continui → misure → si usano delle soglie



I meccanismi automatici sono usati per costruire modelli compatti, ovvero con meno test possibili, secondo il principio del rasoio di Occam.



scegliere l'attributo più promettente per la costruzione di split significativi

La confusione di un insieme si misura tramite:

- Entropia
- Gini
- Errore di classificazione

Sia  $P(i|t)$ : probabilità che un elemento in  $t$  sia di classe  $i$

$$P(i|t) \in [0,1]$$

$$\text{ENTROPIA}(t) = - \sum_{i=0}^{C-1} P(i|t) \cdot \log_2 P(i|t) \quad C = \# \text{ classi}$$



**GUADAGNO:**  $\Delta = I(\text{PARENT}) - \sum_{j=1}^k \left[ \underbrace{\frac{N(v_j)}{N}}_{\text{PESO}} \cdot I(v_j) \right]$

(information gain)



$\Delta$  = impurità

calcolata  
con l'entropia

$k = \# \text{ figli}$

$N = \# \text{ istanze del Learning Set}$   
associate al nodo parent

$\Delta = \text{confusione del parent}$

$N(v_j) = \# \text{ istanze del L.S.}$   
associate a PARENT  
per cui l'attributo  
vale  $v_j$

ridotta di una q.ta'

(sottoinsieme del L.S.)

che dipende dalla

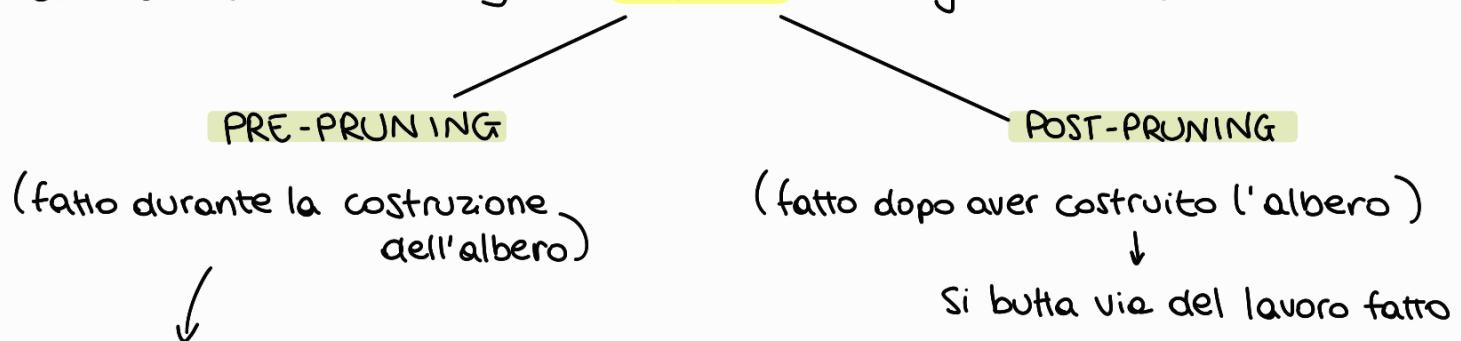
confusione dei nodi figli

**OVERFITTING** : errore di generalizzazione

- ↳ L.S. troppo specializzato
- ↳ L.S. contiene errori

Più un modello è preciso sul learning set, maggiore è il rischio di overfitting

Per ridurre l'overfitting: **PRUNING** → tagliare rami dell'albero



Sia  $D_t$  la cardinalità di un nodo

si utilizza una soglia  $s = \# \text{min}$  di istanze t.c.

se  $D_{t(\text{child})} < s \Rightarrow$  non viene fatto lo split e il nodo diventa foglia

L'algoritmo non è la parte più importante dell'apprendimento automatico :

1) Dati → avere un buon Learning Set

2) Valutazione del modello

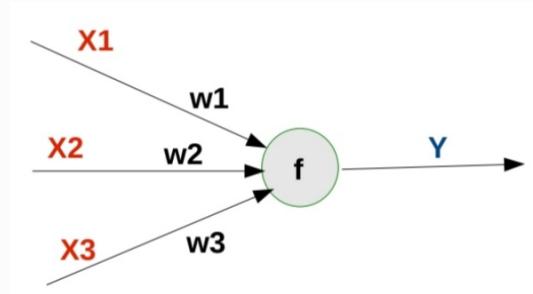
Alberi di decisione: più adatti per problemi di natura simbolica

Problemi di natura numerica → reti neurali

## RETI NEURALI

Perceptron = modello matematico del neurone

- elemento computazionale dotato di una piccola memoria, che calcola una funzione in esso strettamente codificata



$$Y = f(\text{net})$$

FUNZIONE DI ATTIVAZIONE

calcolata su una combinazione dei valori in input, opportunamente pesati

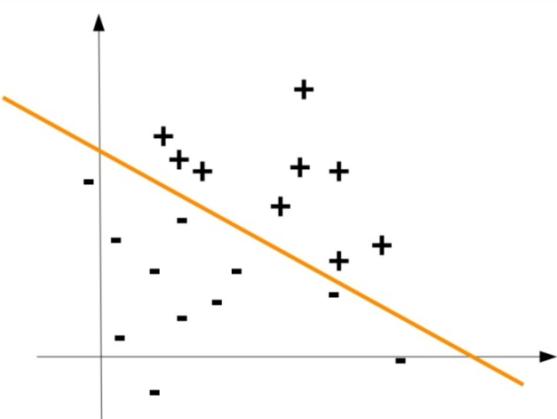
$$\text{net} = \sum_{i=1}^n w_i \cdot x_i$$

- Originariamente  $f$  era la funzione gradino  $\rightarrow f(\text{net}) = \begin{cases} 0 \\ 1 \end{cases}$   
↳ non derivabile
- $f$  diventa una funzione sigmoide  $\rightarrow$  derivabile



$\theta$  : soglia (preimpostata)  
o bias

Con il perceptron si cattura un test lineare



- Cio' che cade al di sopra dell' iperpiano codificato dai suoi pesi fa attivare il neurone
- sopra : appartenente alla classe obiettivo
- sotto : istanze negative

**PESI**: caratterizzano i neuroni → costituiscono la conoscenza del neurone

⇒ Apprendimento del perceptron = trovare i pesi giusti per costruire il test lineare giusto per risolvere un certo problema

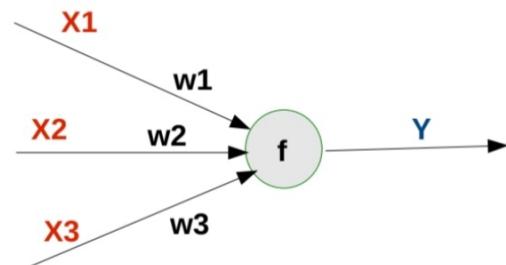
Perceptron ha bisogno di un learning set supervisionato

I <sub>1</sub>	c <sub>1</sub>
I <sub>2</sub>	c <sub>2</sub>
I <sub>3</sub>	c <sub>3</sub>
...	...

### APPRENDIMENTO:

- 1) Inizializzazione casuale dei pesi
- 2) Passata Forward → prendere una tupla di valori del L.S. come input e viene prodotto un output  $\circ$ .  
Sia  $d$  l'output desiderato:  $(d - o)$  = errore
- 3) Passata Backward → Si prende l'errore e lo usa per modificare i pesi secondo le formule:  
 $w_j^{(k+1)} = w_j^k + \alpha \cdot (d - o) x_j$
- 4) Si passa all'istanza successiva

EPOCA DI APPRENDIMENTO  
= Singola passata del L.S.



Bisogna rielaborare l'esperienza per molte epochhe di apprendimento per raggiungere la convergenza.

**LIMITE**: Un singolo perceptron non basta

↳ dimostrato che non puo' imparare lo **XOR**



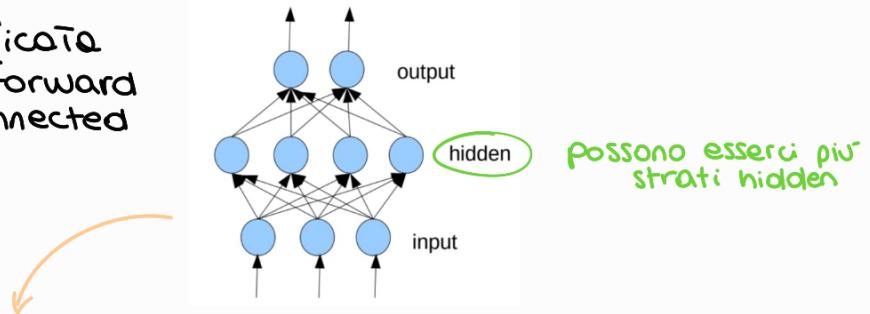
**Soluzione**: combino l'effetto di piu' perceptron

Rete neurale = combinazione di neuroni artificiali

- Diverse tipologie in base a:
  - funzioni calcolate dai singoli neuroni
  - topologia della rete
  - modello di apprendimento
- Multi Layer Perceptron (MLP)

Topologia:

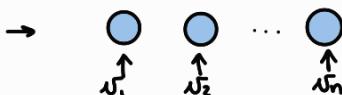
- stratificata
- feed forward
- full connected



- Strato di input:

tanti neuroni quanti sono gli attributi descrittivi dell'istanza

$$i = \langle n_1, n_2, \dots, n_n \rangle$$



↳ non è costituito da perceptron

- Strato hidden:

- l'istanza viene propagata da ciascun neurone input a ciascuno del primo strato hidden
- il valore calcolato nel primo strato viene propagato ai neuroni degli strati successivi (se ci sono)
- Sono dei perceptron

- Strato di output:

- i neuroni combinano i risultati dei neuroni hidden

### CLASSIFICAZIONE (numero di output)

- se Problema = riconoscere le istanze della classe X → 1 neurone di output
- se Problema = distinguere le istanze di classe X da quelle di classe Y → 1 neurone di output
- se Problema = distinguere fra 3 classi → 2 neuroni (2 bit per rappresentare il numero 3)

Spesso si usa un neurone di output per ogni classe.

Il neurone corrispondente alla classe giusta deve attivarsi, gli altri devono rimanere a 0

## APPRENDIMENTO

- impara in modo supervisionato → Learning set

- per ogni istanza :

• PASSATA FORWARD

• PASSATA BACKWARD

} come già visto nei perceptron

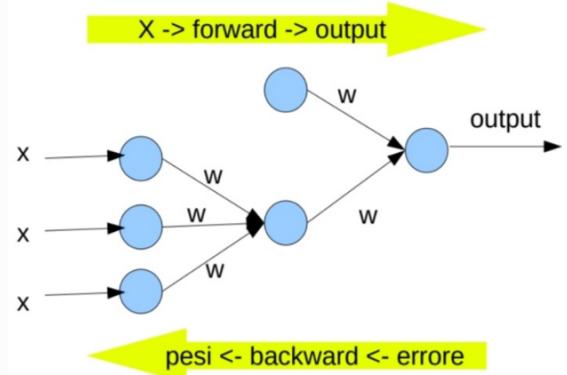
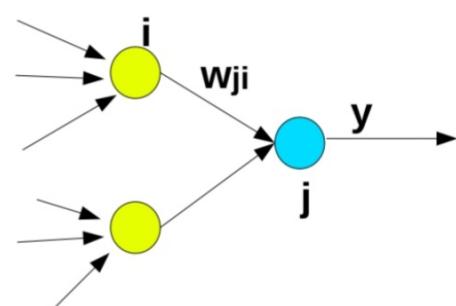
l'errore  $E$  dipende dalla matrice dei pesi  $W$

→ distribuire l'errore e aggiornare i pesi

## Discesa del gradiente

Vogliamo trovare un modo per modificare i pesi che riduca l'errore

$$\Delta w_{ji} = -\lambda \frac{\partial E(\bar{w})}{\partial w_{ji}}$$



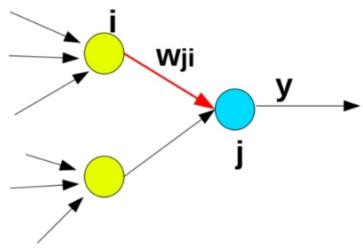
## ERRORE GLOBALE

$$E = \frac{P}{2} \sum_{i=1}^l \|t_i - y_i\|^2$$

P: #neuroni di output

$t_i$ : valore desiderato per il neurone di output  $i$ -mo

$y_i$ : valore prodotto dal neurone di output  $i$ -mo



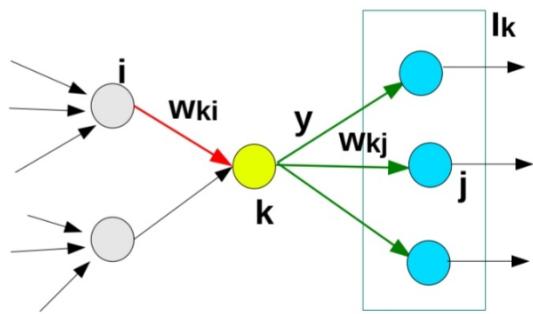
## BACKPROPAGATION

Delta Rule

$$\Delta w_{ji} = \alpha \cdot \delta^j \cdot x_{ji}$$

$$\delta^j = y_j \cdot (1 - y_j) \cdot (t_j - y_j)$$

modifica i pesi  
dello strato output  
strati hidden?



$$\Delta w_{ki} = \alpha \cdot \delta^k \cdot x_{ki}$$

$$\delta^k = y \cdot (1 - y) \cdot \sum_{j \in I_k} \delta^j w_{kj}$$

misura derivante dalla  
retro-propagazione dell'errore

## REINFORCEMENT LEARNING

↳ imparare un comportamento per esperienza, sperimentando.

CONCETTI CHIAVE:

- **Condizionamento** : modifica il comportamento riflesso, non controllato in maniera consci.
- **Condizionamento operante** : modifica il comportamento consci grazie a un meccanismo premi/punizioni

Gli agenti SW agiscono in un ambiente:

→ posso causare la costruzione automatica di un "comportamento" introducendo una forma di condizionamento operante nei sistemi SW

- **Policy** : strategia di scelta delle operazioni da eseguire

- **Compenso atteso** :  $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}$   $0 \leq \gamma \leq 1$

- **Processo di decisione Markoviano (MDP)**

↳ lo stato è indipendente dal cammino

$$P_{ss'}^a = \Pr \{ S_{t+1} = s' \mid S_t = s, a_t = a \} \quad \text{PROBABILITÀ DI TRANSIZIONE}$$

$$R_{ss'}^a = E \{ r_{t+1} \mid S_t = s, a_t = a, S_{t+1} = s' \} \quad \text{VALORE ATTESO DEL PROSSIMO REWARD}$$

- **Mapping** :  $\Pi_t$  tabella delle probabilità



Mapping:

$s_1 \rightarrow p_{a1} p_{a2} \dots p_{an}$   
 $s_2 \rightarrow p_{a1} p_{a2} \dots p_{an}$   
...  
 $s_n \rightarrow p_{a1} p_{a2} \dots p_{an}$

$\Pi_t(s, a)$ : probabilità di selezionare l'azione a essendo nella situazione s

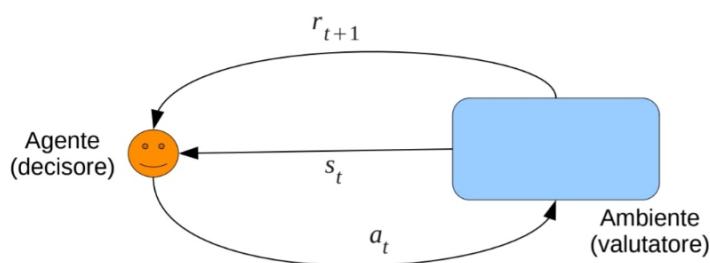
**Apprendimento per rinforzo**: approccio computazionale all'apprendimento per interazione



→ imparare cosa fare **associando azioni a situazioni** in modo tale da massimizzare un segnale numerico di compenso

Processo **Trial-and-error** → l'agente costruisce il **Controllore** in autonomia  
↓  
**programma che mappa i dati sensoriali**

→ il programmatore deve scrivere la **funzione di rinforzo**, dopo aver identificato gli stati di successo e quelli di fallimento



L'interazione avviene ad ogni istante di una sequenza temporale  $t = 1, 2, 3, \dots$

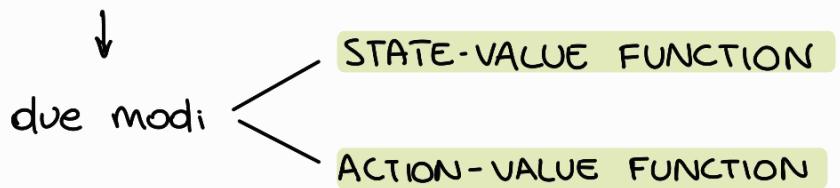
- Ad ogni istante, l'agente:
  - 1) Riceve una rappresentazione della situazione corrente ( $s_t$ )
  - 2) Sceglie un'azione  $a_t$  fra quelle eseguibili e la esegue
- All'istante successivo, l'agente riceve un **rinforzo numerico** ( $r_{t+1}$ ) e tutto ricomincia

Risolvere un problema di Reinforcement Learning

≡ costruire una **Policy** che permette di ottenere compensi alti sul lungo termine

**Goal dell'agente** → massimizzare il compenso ricevuto complessivamente  
indica COSA si desidera  
che l'agente faccia, non COME

Usare la conoscenza → funzione di valutazione degli stati



### STATE-VALUE FUNCTION

**POLICY**

$$V^\pi(s) = \mathbb{E} \{ R_t | s_t = s \} = \frac{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s}{\text{aspettativa di compenso futuro se lo stato corrente è } s}$$

**compenso atteso**  
 $\gamma \in [0,1]$

### ACTION-VALUE FUNCTION

$$Q^\pi(s,a) = \mathbb{E}_\pi \{ R_t | s_t = s, a_t = a \}$$

Aspettativa di compenso futuro  
se s è lo stato corrente e in s  
si esegue a

**PROBLEMA :** approssimare la funzione di valutazione

Proprietà fondamentale delle funzioni di valutazione:

Equazione di Bellman

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')]$$

Problemi di RL :

- di **predizione** → costruire la valutazione di una policy
- di **controllo** → costruire una policy

Alcuni metodi di risoluzione

- Monte Carlo → problemi di natura episodica
- Temporal Difference (state-value function)
- Q-learning (action-value function)

## Temporal difference

TD(0) → problema di predizione

Siano:

$\pi$ : la policy da valutare

$V$ : una funzione di valutazione arbitraria

Repeat per ogni episodio:

Inizializza  $s$

Repeat per ogni passo  $s$  dell'episodio:

(1)  $a :=$  azione restituita da  $\pi$  per  $s$

(2) esegui  $a$

(3) osserva il rinforzo immediato  $r$  e lo stato successivo  $s'$

(4)  $V(s) := V(s) + \alpha [r + \gamma V(s') - V(s)]$

(5)  $s := s'$

end

end

N.B. l'insieme degli stati è predefinito