

# IA

Silviu Filote

July 2023

## Contents

<b>1</b>	<b>Introduzione</b>	<b>1</b>
<b>2</b>	<b>Logica proposizionale</b>	<b>1</b>
2.1	Sintassi . . . . .	1
2.2	Semantica . . . . .	1
2.3	Sistemi di prova . . . . .	3
2.4	Clausole di Horn . . . . .	6
<b>3</b>	<b>Logica del prim'ordine</b>	<b>7</b>
3.1	Quantificatori e forme normali . . . . .	8
3.2	Risoluzione nella logica del prim'ordine . . . . .	9
<b>4</b>	<b>Risoluzione Logica</b>	<b>11</b>
4.1	Dimostrazione per inferenza . . . . .	12
4.2	Dimostrazione per contraddizione + Risoluzione . . . . .	12
<b>5</b>	<b>PROLOG</b>	<b>13</b>
5.1	Sintassi . . . . .	13
5.2	Esempio . . . . .	13
5.3	Funzionamento di PROLOG . . . . .	14
5.4	Backtracking . . . . .	15
5.5	Risoluzione query: . . . . .	16
5.6	Distinzione tra semantica dichiarativa e procedurale in PROLOG . . . . .	17
5.7	Elementi procedurali e controllo dell'esecuzione . . . . .	17
5.8	Liste . . . . .	18
<b>6</b>	<b>Search</b>	<b>19</b>
6.1	Definizioni . . . . .	19
6.2	Esempio . . . . .	20
6.3	Esempio 2 . . . . .	21
6.4	Uninformed search . . . . .	21
6.5	Breadth-first search . . . . .	21
6.6	Depth-first search . . . . .	22
6.7	Approfondimento iterativo (iterative deepening) . . . . .	23
6.8	Controllo di cicli (cycle check) . . . . .	23
<b>7</b>	<b>Algoritmi di ricerca</b>	<b>24</b>
7.1	Heuristic breadth-first . . . . .	24
7.2	Euristica e stime . . . . .	24
7.3	Problema semplificato: esempio . . . . .	25
7.4	Algoritmo A* . . . . .	26
7.5	Esempio: applicazione A* . . . . .	27
7.6	Ricerca con avversari . . . . .	27
7.7	Gioco a singolo turno . . . . .	28
7.8	Algoritmo MINIMAX . . . . .	28
7.9	Potatura . . . . .	30
7.10	Giochi non deterministici . . . . .	33

<b>8 Constraint Satisfaction problem</b>	<b>35</b>
8.1 Definizione del problema . . . . .	35
8.2 Varianti del CSP . . . . .	36
8.3 Inferenza nei CSP . . . . .	36
8.4 Algoritmo AC3 . . . . .	37
8.5 Esempio: Sudoku . . . . .	38
8.6 Search in CSP . . . . .	38
8.7 Esempio: mappa Australia . . . . .	39
8.8 Strategie . . . . .	39
8.9 Forward checking (FC) . . . . .	40
8.10 Backjumping e conflict set . . . . .	41
8.11 Un altro approccio: ricerca locale . . . . .	42
<b>9 Apprendimento automatico</b>	<b>43</b>
9.1 Machine Learning e Data Mining . . . . .	43
9.2 Analisi dei dati . . . . .	44
9.3 Il percettrone: un classificatore lineare . . . . .	45
9.4 Il metodo Nearest Neighbor (il vicino più vicino) . . . . .	47
9.5 La classificazione in più classi . . . . .	48
9.6 Confronto . . . . .	48
9.7 Case Based Reasoning (CBR) . . . . .	49
9.8 Decision Tree Learning . . . . .	49
9.9 Entropia come misura di informazione . . . . .	50
9.10 Potatura dell'albero . . . . .	52
9.11 Cross-validation . . . . .	52
9.12 One-class Learning . . . . .	53
9.13 Nearest Neighbor Data Description . . . . .	53
9.14 Clustering . . . . .	53
9.15 k-Means e l'algoritmo EM . . . . .	54
9.16 Clustering gerarchico . . . . .	55
9.17 Come determinare il numero di cluster? . . . . .	55
<b>10 Reti Neurali</b>	<b>57</b>
10.1 Il modello matematico . . . . .	57
10.2 Reti di Hopfield . . . . .	58
10.3 Memorie associative neurali: esempio riconoscimento facciale . . . . .	59
10.4 Memoria con matrice di correlazione . . . . .	59
10.5 Reti lineari con errori minimi . . . . .	60
10.6 Metodo dei minimi quadrati . . . . .	61
10.7 La regola del Delta . . . . .	61
10.8 Reti neurali e backpropagation . . . . .	62
10.9 Le 4 equazioni fondamentali della backpropagation . . . . .	64
<b>11 Dimostrazioni per esame</b>	<b>67</b>

# 1 Introduzione

Che cos'è l'intelligenza artificiale?

- L'obiettivo dell'IA è sviluppare macchine che si comportino come se fossero intelligenti. *John McCarthy, 1955*
- L'intelligenza artificiale è la capacità dei computer digitali o dei robot controllati da computer di risolvere problemi normalmente associati alle capacità di elaborazione intellettuale più elevate degli esseri umani. *Enciclopedia Britannica*
- L'intelligenza artificiale è lo studio di come far fare ai computer cose in cui, al momento, le persone sono migliori. *Elaine Rich, UT Austin*

I problemi del Machine Learning: algoritmi black-box e crea disoccupazione.

Alan turiring gioco dell'imitazione:

- **standard** C (umano) deve capire chi é il computer e chi é l'umano
- **originale**

## 2 Logica proposizionale

Nella logica proposizionale, come suggerisce il nome, proposizioni sono collegate da operatori logici. Questa notazione ha il vantaggio che le proposizioni elementari appaiono in forma inalterata.

*se piove la strada é bagnata  
piove  $\Rightarrow$  la strada é bagnata*

### 2.1 Sintassi

**Definizione:** siano  $OP = \{\neg; \wedge; \vee; \Rightarrow; \Leftrightarrow; (\cdot)\}$  l'**insieme di operatori logici** e  $\Sigma$  un insieme di simboli. Gli insiemi  $OP, \Sigma$  e  $\{t, f\}$  (true, false) sono disgiunti a due a due (non hanno elementi in comune).  $\Sigma$  è chiamata la **firma (signature)** e i suoi elementi sono le **variabili proposizionali**. L'insieme delle formule della logica proposizionale è ora definito in modo ricorsivo:

- $t$  e  $f$  sono **formule (atomiche)**
- Tutte le variabili proposizionali, cioè tutti gli elementi di  $\Sigma$  sono **formule atomiche**
- se  $A$  e  $B$  sono formule, allora  $\neg A$ ,  $(A)$ ,  $A \wedge B$ ,  $A \wedge B$ ,  $A \Rightarrow B$ ,  $A \Leftrightarrow B$  sono **formule atomiche**

I simboli e gli operatori si leggono come segue:

$t$	<i>vero</i>
$f$	<i>falso</i>
$\neg A$	<i>non A (negazione)</i>
$A \wedge B$	<i>A e B (congiunzione)</i>
$A \vee B$	<i>A o B (disgiunzione)</i>
$A \Rightarrow B$	<i>se A allora B (condizionale)</i>
$A \Leftrightarrow B$	<i>A se e solo se B (doppio condizionale)</i>

### 2.2 Semantica

Nella logica proposizionale ci sono due valori di verità:  $t$  e  $f$ . Dobbiamo ovviamente assegnare valori di verità che riflettono lo stato del mondo alle variabili di proposizione.

**Definizione:** un assegnamento  $I : \Sigma \rightarrow \{t; f\}$ , che assegna un valore di verità a ogni simbolo, si chiama **interpretazione**.

Poiché ogni variabile proposizionale può assumere due valori di verità, *ogni formula proposizionale con n variabili differenti ha  $2^n$  interpretazioni differenti*. Definiamo i valori di verità per le operazioni di base mostrando tutte le possibili interpretazioni in una cosiddetta **tavola della verità**:

$A$	$B$	$(A)$	$\neg A$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftrightarrow B$
t	t	t	f	t	t	t	t
t	f	t	f	f	t	f	f
f	t	f	t	f	t	t	f
f	f	f	t	f	f	t	t

Osservazioni:

- La formula vuota è vera per tutte le interpretazioni;
- Se le espressioni sono tra parentesi, il termine tra parentesi viene valutato per primo;
- Per le formule senza parentesi, le priorità sono ordinate come segue, a partire dal legame più forte:

$$\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$$

**Definizione:** due formule  $F$  e  $G$  sono chiamate (**semanticamente**) **equivalenti** se assumono lo stesso valore di verità per tutte le interpretazioni. Scriviamo  $F \equiv G$

L'equivalenza semantica permette di ragionare sul linguaggio oggetto (la logica) utilizzando il meta-linguaggio (il linguaggio naturale).

$$A \equiv B$$

*confrontare il significato di due formule  
o espresini nel linguaggio naturale  $A$  e  $B$  sono  
semanticamente equivalenti*

*confrontare il significato di due formule proposizionali  
nel linguaggio formale.  $A$  e  $B$  sono logicamente  
equivalenti*

In sintesi, l'equivalenza semantica è una nozione utilizzata per confrontare il significato di due formule nel linguaggio naturale, mentre l'operatore logico di equivalenza ( $\Leftrightarrow$ ) è un elemento del linguaggio formale della logica proposizionale utilizzato per rappresentare l'equivalenza logica tra due formule.

**Definizione:** Una formula si dice:

- **soddisfacibile** e se è vera per almeno un'interpretazione (nota anche come **contingenza**).
- (**logicamente**) **valida** o **tautologia** se è vera per tutte le interpretazioni.  $A \vee \neg A$
- **insoddisfacibile** o **contraddizione** se non è vera per alcuna interpretazione.  $A \wedge \neg A$

Ogni interpretazione che soddisfa una formula è chiamata **modello della formula**.

**Teorema:** Gli operatori  $\wedge$  e  $\vee$  sono commutativi e associativi e sono valide le seguenti equivalenze:

$$\neg A \vee B \equiv A \Rightarrow B$$

$$A \Rightarrow B \equiv \neg B \Rightarrow \neg A$$

*legge di contrapposizione*

$$(A \Rightarrow B) \wedge (B \Rightarrow A) \equiv (A \Leftrightarrow B)$$

*legge di DeMorgan 1*

$$\neg(A \wedge B) \equiv \neg A \vee \neg B$$

*legge di DeMorgan 2*

$$\neg(A \vee B) \equiv \neg A \wedge \neg B$$

*distributività di  $\vee$  su  $\wedge$*

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$

*distributività di  $\wedge$  su  $\vee$*

$$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$$

*tautologia*

$$A \vee \neg A \equiv t$$

*contraddizione*

$$A \wedge \neg A \equiv f$$

$$A \vee f \equiv A$$

$$A \vee t \equiv t$$

$$A \wedge f \equiv f$$

$$A \wedge t \equiv A$$

*Dimostrazione tavola di verità di:  $\neg A \vee B \equiv A \Rightarrow B$*

$A$	$B$	$\neg A$	$\neg A \vee B$	$A \Rightarrow B$	$(\neg A \vee B) \Leftrightarrow (A \Rightarrow B)$
t	t	f	t	t	t
t	f	f	f	f	t
f	t	t	t	t	t
f	f	t	t	t	t

## 2.3 Sistemi di prova

Nell'intelligenza artificiale siamo interessati a prendere conoscenza preesistente, e da quella derivare nuova conoscenza, o usarla per rispondere a domande. Nella logica proposizionale questo significa mostrare che da una base di conoscenza  **$KB$**  segue una formula  **$Q$** .

- **$KB$  (Knowledge Base):** una formula di logica proposizionale (ossia un certo numero di formule messe in congiunzione tra loro)
- **$Q$  (Query):** chiamata anche tesi in ambito della logica

**Definizione:** Una formula  $KB$  **implica** una formula  $Q$  (o  $Q$  **segue da**  $KB$ ) (o  $Q$  è una **conseguenza semantica di**  $KB$ ) se ogni modello di  $KB$  è anche un modello di  $Q$ . Scriviamo dunque:

$$KB \models Q \quad \text{"implicazione"}$$

- In ogni interpretazione in cui  $KB$  è vera, anche  $Q$  è vera.
- Ogni formula non valida (ossia non sempre vera) sceglie, per così dire, un sottoinsieme dell'insieme di tutte le interpretazioni come proprio modello
- La formula vuota è vera in tutte le interpretazioni. Per ogni tautologia  $T$  allora:  $\emptyset \models T$ . Questo significa che le tautologie sono sempre vere, senza restrizione delle interpretazioni di una formula. In breve scriviamo  $\models T$ .

$$\models T$$

*Non abbiamo bisogno di nessuna premessa  
perché  $T$  risulta essere sempre vera  $\Rightarrow T$  è una tautologia*

**Teorema di deduzione:**

$$A \models B \quad \text{se e solo se} \quad \models A \Rightarrow B \\ \models A \Rightarrow B \text{ vuol dire che } A \Rightarrow B \text{ è una tautologia}$$

*Dimostrazione :*

A	B	$A \Rightarrow B$
t	t	t
t	f	f
f	t	t
f	f	t

- **Dimostrazione da sx a dx:** dunque supponiamo che  $A \models B$  valga. Ciò significa che per ogni interpretazione che rende  $A$  vero, anche  $B$  è vero. Si considera solo la 1° riga e la 2° riga si scarta. Dunque  $A \Rightarrow B$  è vero, il che significa che  $A \Rightarrow B$  è una tautologia.
- **Dimostrazione da dx a sx:** supponiamo che  $A \Rightarrow B$  sia una tautologia. Ogni modello di  $A$  è allora anche il modello di  $B$ , dunque  $A \models B$ .

**Osservazioni importanti:**

- Se vogliamo dimostrare che  $KB$  implica  $Q$ , possiamo anche dimostrare mediante il metodo della tavola di verità che:

$$KB \Rightarrow Q \quad \text{è una tautologia}$$

- Con questo metodo abbiamo il primo sistema di prova per la logica proposizionale automatizzabile
- Lo svantaggio di questo metodo è il tempo di calcolo molto lungo, nel peggiore dei casi con  $n$  variabili di proposizione, per tutte le  $2^n$  interpretazioni delle variabili deve essere valutata la formula  $KB \Rightarrow Q$
- **NB:** se  $KB \models Q$ , allora per il teorema di deduzione  $KB \Rightarrow Q$  è una tautologia, ossia:

$$\neg(KB \Rightarrow Q) \quad \text{è insoddisfacibile}$$

$$\neg(KB \Rightarrow Q) \equiv \neg(\neg KB \vee Q) \equiv KB \wedge \neg Q \quad \text{è insoddisfacibile}$$

**Teorema di deduzione (Dimostrazione per contraddizione):**

$$KB \models Q \quad \text{se e solo se} \quad KB \wedge \neg Q \text{ è insoddisfacibile}$$

- Per mostrare che  $KB \models Q$ , possiamo aggiungere  $\neg Q$  alla  $KB$  e derivare una contraddizione
- Sappiamo che  $A \wedge \neg A \equiv f$  è una contraddizione. In un modello di  $KB$ , ossia in una interpretazione in cui  $KB$  è vera, abbiamo quindi che  $KB \wedge \neg Q$  non può essere vera  $\Rightarrow$  quindi si deduce la verità di  $Q$ .

Un modo per evitare di dover testare tutte le interpretazioni con il metodo della tavola di verità è la manipolazione sintattica delle formule  $KB$  e  $Q$  mediante l'applicazione di **regole di inferenza** con l'obiettivo di **semplificarle** notevolmente. Chiamiamo questo processo **derivazione sintattica** e scriviamo:

$$KB \vdash Q$$

**Definizione:** Un calcolo è chiamato **corretto (sound)** se ogni proposizione derivata per via sintattica con il calcolo è una conseguenza semantica. Cioé, vale per le formule  $KB$  e  $Q$  quanto segue:

$$\text{se } KB \vdash Q \quad \text{allora} \quad KB \models Q$$

**Definizione:** Un calcolo è detto **completo** se tutte le conseguenze semantiche possono essere derivate con il calcolo. Cioé, per le formule  $KB$  e  $Q$  vale quanto segue:

$$\text{se } KB \models Q \quad \text{allora} \quad KB \vdash Q$$

Osservazioni:

- La correttezza di un calcolo garantisce che tutte le formule derivate siano di fatto conseguenze semantiche della base di conoscenza  $\rightarrow$  il calcolo non produce **false conseguenze**
- La completezza di un calcolo, d'altra parte, assicura che il calcolo non trascuri nulla.
- Se un calcolo è corretto e completo, la derivazione sintattica e la conseguenza semantica sono due relazioni equivalenti
- Per mantenere i sistemi di prova automatici i più semplici possibili, essi sono solitamente disegnati per operare su formule in **forma normale congiuntiva**.

**Definizione:** Una formula è in **forma normale congiuntiva** (“conjunctive normal form”, CNF) se e solo se consiste in una congiunzione  $K_1 \wedge K_2 \wedge \dots \wedge K_m$  di clausole. Una **clausola**  $K_i$  consiste in una disgiunzione  $(L_{i1} \vee L_{i2} \vee \dots \vee L_{in})$  di letterali. Un **letterale** è una variabile proposizionale ( $A$  letterale positivo) o una variabile proposizionale negata ( $\neg B$  letterale negativo).

$$(A \vee B \vee \neg C) \wedge (A \vee B) \wedge (\neg B \vee \neg C) \quad \text{CNF}$$

**Teorema:** Ogni formula della logica proposizionale può essere trasformata in una forma normale congiuntiva equivalente.

Il modo di procedere più comune è:

1. eliminazione  $\Leftrightarrow$  trasformandolo in congiunzione di  $\Rightarrow$
2. eliminazione di  $\Rightarrow$  trasformandolo in negazione e disgiunzione
3. De Morgan per spostare le negazioni “verso l'interno”, in modo da avere letterali negativi
4. applicare distributività e associatività per ottenere congiunzioni di disgiunzioni

**Regola di inferenza:** una regola che manipola le formule per ottenere delle altre chiamate **derivate**.

**Modus ponens (regola di inferenza):** dalla validità di  $A$  e  $A \Rightarrow B$  consente la derivazione di  $B$ . In maniera più informale:

*se sono vere:  $A$  e  $A$  implica  $B$ , allora possiamo concludere che  $B$  è vera.*

$$\frac{\text{Input/premesse}}{\text{Output/risultato}} \quad \frac{A, \ A \Rightarrow B}{B}$$

Questa notazione significa che possiamo derivare le formule al denominatore impiegando quelle al numeratore. Il modus ponens da solo, come regola di inferenza, è corretto ma non è completo.

**Modus ponens:**  
*corretto, ma non completo*

$$\frac{A}{A \vee B} \rightarrow \text{impossibile}$$

**Risoluzione (regola di inferenza):**

- Le formule in input, le **premesse** sono **clausole**, ossia disgiunzioni di letterali. (sono sempre vere)
- Da una parte e dall'altra rispetto ai 2 input ci sia almeno una lettera ripetuta e deve comparire da una parte negata e dall'altra no.
- La formula derivata si chiama **risolvente**, ossia l'output.
- La regola di risoluzione elimina una coppia di letterali complementari dalle due clausole e combina il resto dei letterali in una nuova clausola.

*Entrambe le premesse sono vere:  $(A_1 \vee \dots \vee A_m \vee B)$  e  $(\neg B \vee C_1 \vee \dots \vee C_n)$   
allora possiamo concludere che  $(A_1 \vee \dots \vee A_m \vee C_1 \vee \dots \vee C_n)$  è vera  
 $B$  e  $\neg B$  sono complementari*

$$\frac{(A_1 \vee \dots \vee A_m \vee B), (\neg B \vee C_1 \vee \dots \vee C_n)}{(A_1 \vee \dots \vee A_m \vee C_1 \vee \dots \vee C_n)}$$

**Teorema:** Il calcolo della *risoluzione* per la **dimostrazione di insoddisfacibilità** di formule in forma congiuntiva normale è valido e completo.

Se è completo, allora dovrebbe riuscire a derivare  $A \vee B$  da  $A$ . Visto che la dimostrazione è di insoddisfacibilità, noi usiamo la negazione della formula che vogliamo derivare, la aggiungiamo alla  $KB$  e procediamo con la risoluzione, in cerca della clausola vuota.

**Risoluzione:**  
*corretto, e completo*

$$\begin{array}{ll} \text{Tesi :} & A \vee B \\ \text{KB :} & A \\ \text{Tesi negata (TN):} & \neg(A \vee B) \equiv \neg A \wedge \neg B \\ KB' = KB \cup \{TN\} : & \{A, \neg A, \neg B\} \end{array}$$

$$\frac{A, \ \neg A}{\{\}}$$

- Le formule in CNF vengono separate in clausole nella  $KB$
- Da  $A$  e  $\neg A$  con la risoluzione ottengo subito la **clausola vuota**  $\{\}$ , dunque nella *Tesi + KB* c'è una contraddizione
- Quindi la tesi originale (quella non negata) è sempre vera quando la  $KB$  è vera, ossia è una sua conseguenza semantica

**Definizione:** Una formula  $KB$  si dice **consistente** se non è possibile derivarne una contraddizione, cioè una formula della forma  $\emptyset \wedge \neg\emptyset$ .

## 2.4 Clausole di Horn

Una clausola in forma normale congiuntiva contiene letterali positivi e negativi e può essere rappresentata nella forma:

$$(\neg A_1 \vee \dots \vee \neg A_m \vee B_1 \vee \dots \vee B_n)$$

con tutti i letterali negativi a sinistra e quelli positivi a destra. Usando l'equivalenza  $\neg A \vee B \equiv A \Rightarrow B$  possiamo trasformare la clausola nella formula:

$$\begin{aligned} & \neg(A_1 \wedge A_m) \\ & (A_1 \wedge \dots \wedge A_m) \Rightarrow (B_1 \vee \dots \vee B_n) \end{aligned}$$

*Frase che riassume:*

*"Se il tempo è bello e c'è neve, allora andrò a sciare oppure andrò in ufficio."*

Non una frase particolarmente informativa. Molto più chiara è una frase come questa:

$$(A_1 \wedge \dots \wedge A_m) \Rightarrow B_1$$

*"Se il tempo è bello e c'è neve e il mio capo è d'accordo, allora andrò a sciare."*

*In forma a clausole:*

$$\neg A_1 \vee \dots \vee \neg A_m \vee B_1$$

Questa è una clausola con un solo letterale positivo. Questo tipo di clausole ha il vantaggio di **consentire solo una conclusione** e sono quindi decisamente più semplici da interpretare.

Clausole con al massimo un letterale positivo si chiamano **clausole definite o clausole di Horn**

**Definizione:** Clausole con al massimo un letterale positivo nella forma:

$$\neg A_1 \vee \dots \vee \neg A_m \vee B \quad \text{oppure} \quad \neg A_1 \vee \dots \vee \neg A_m \quad \text{oppure} \quad B$$

*o, equivalentemente*

$$(\neg A_1 \wedge \dots \wedge \neg A_m) \Rightarrow B \quad \text{oppure} \quad \neg A_1 \wedge \dots \wedge \neg A_m \Rightarrow f \quad \text{oppure} \quad B$$

sono chiamate **clausole di Horn**. Una clausola con un singolo letterale positivo è chiamata **fatto**. Nelle clausole con letterali negativi e un letterale positivo, il letterale positivo è chiamato **testa**.

**Osservazione:**

- Nel caso di grandi basi di conoscenza, tuttavia, il modus ponens può derivare molte formule non necessarie se si inizia con le clausole sbagliate.
- Pertanto, in molti casi è meglio utilizzare un calcolo che inizi con la query e funzioni a ritroso fino a raggiungere i fatti. Tali sistemi sono chiamati **backward chaining** (concatenamento all'indietro), in contrasto con i sistemi di **forward chaining** (concatenamento in avanti), che iniziano con i fatti e infine derivano la tesi.
- Per il concatenamento all'indietro delle clausole Horn, viene utilizzata la risoluzione SLD. **SLD** sta per **“Selection-rule-driven Linear resolution for Definite clauses”** (**risoluzione lineare basata su regole di selezione per clausole definite**).
- Risoluzione lineare: il passo successivo viene sempre eseguito sulla clausola appena derivata. Questo porta ad una grande riduzione dello spazio di ricerca.

### 3 Logica del prim'ordine

$V$	un insieme di variabili: $\{x, y, z, a, b, \dots\}$	start lettera minuscola
$K$	un insieme di costanti: $\{1, 2, Bob, Cake, Car\}$	start lettera maiuscola
$F$	un insieme di simboli funzionali: $\{sqrt, sum, min, likes\}$	start lettera minuscola
$P$	l'insieme dei simboli di predicato: $\{Brown, Frogs, Red\}$	start lettera maiuscola

**Definizione:** definiamo l'**insieme dei termini** come:

- tutte le variabili e tutte le costanti sono termini;
- se  $t_1, t_2, \dots, t_k$  sono termini e  $f$  è un simbolo funzionale a  $k$  posti, allora  $f(t_1, t_2, \dots, t_k)$  è un termine.

In più, sia  $P$  l'insieme dei **simboli di predicato**, con ciascun simbolo caratterizzato da una **aritá**, ossia il numero di termini a cui il predicato si applica.

**Definizione:** Definizione delle formule della logica del prim'ordine:

- se  $t_1, \dots, t_n$  sono  $n$  termini e  $P$  è un simbolo di predicato  $n$ -ario, allora  $P(t_1, \dots, t_n)$  è una **formula**
- se  $A$  e  $B$  sono formule, allora lo sono anche  $\neg A, A \vee B, A \wedge B, A \Rightarrow B, A \Leftrightarrow B$
- se  $x$  è una variabile e  $A$  è una formula, allora anche  $\forall x A$  e  $\exists x A$  lo sono, con  $\forall, \exists$  **quantificatori**

Per semantica intendiamo il significato della formula, ossia quando essa è vera o falsa.

**Definizione:** una **interpretazione** è

- una corrispondenza tra l'insieme unione delle variabili e le costanti e un insieme di nomi di oggetti nel mondo
- una corrispondenza tra l'insieme dei simboli funzionali e l'insieme delle funzioni nel mondo (con le aritá rispettate)
- una corrispondenza tra l'insieme dei simboli di predicato e le relazioni nel mondo (con le aritá rispettate).

Osservazioni:

- Una formula elementare  $P(t)$  è vera in una interpretazione  $I$  quando a  $t$  corrisponde un oggetto nel mondo che gode della proprietà corrispondente a  $P$
- Una formula  $\forall x A$  è vera in una interpretazione  $I$  quando  $A$  è vera per ogni interpretazione  $I'$  che è identica a  $I$  tranne che per la corrispondenza di  $x$  (sostituisco ad  $x$  un oggetto)
- Una formula  $\exists x A$  è vera in una interpretazione  $I$  quando esiste una interpretazione  $I'$  che è identica a  $I$  tranne che per la corrispondenza di  $x$  in cui  $A$  è vera.
- In sintesi,  $\forall x A$  richiede che  $A$  sia vera per tutte le sostituzioni di  $x$ , mentre  $\exists x A$  richiede che  $A$  sia vera per almeno una sostituzione di  $x$ .

**Teorema:** I teoremi **teorema della deduzione e dimostrazione per contraddizione**, così come le definizioni di **forma normale congiuntiva (CNF)** e di **clausole di Horn** valgono in modo analogo per la logica del prim'ordine.

**Definizione:** Le formule in cui ogni variabile rientra nell'ambito di un quantificatore sono chiamate **formule chiuse**. Le variabili che non rientrano nell'ambito di un quantificatore sono chiamate **variabili libere**.

$$\begin{aligned} \forall x(P(x) \Leftrightarrow Q(x)) & \text{ è una formula chiusa} \\ \forall x \left( \left( R(x) \Rightarrow Q(x) \right) \wedge T(y) \right) & \text{ non è una formula chiusa, } y \text{ è una variabile libera.} \end{aligned}$$

**Definizione:** scriviamo  $\varphi[x/t]$  per indicare la formula che risulta quando **sostituiamo** ogni ricorrenza libera della variabile  $x$  nella formula  $\varphi$  con il termine  $t$ . Il termine non può contenere alcuna variabile che sia quantificata in  $\varphi$ .

$$\begin{aligned} A &= \forall x(x = y) \quad y \text{ variabile libera} \\ A[y/x + 1] &= \forall x(x = x + 1) \quad \text{sostituzione illegittima} \\ A[y/y + 1] &= \forall x(x = y + 1) \quad \text{sostituzione legittima} \end{aligned}$$

**NB:** Una sostituzione in cui una variabile viene sostituita da un'altra variabile  $[x/y]$  si chiama anche **ridenominazione**.

### 3.1 Quantificatori e forme normali

Per tutte le costanti  $A_1, \dots, A_n$  nell'insieme delle costanti  $K$ , possiamo riscrivere i quantificatori come:

$$\begin{array}{ll} \forall x P(x) & P(A_1) \wedge \dots \wedge P(A_n) \\ \exists x P(x) & P(A_1) \vee \dots \vee P(A_n) \\ \\ P(A_1) \wedge \dots \wedge P(A_n) & \equiv \neg\neg(P(A_1) \wedge \dots \wedge P(A_n)) \\ & \equiv \neg(\neg P(A_1) \vee \dots \vee \neg P(A_n)) \\ \\ \forall x P(x) & \equiv \neg\exists x \neg P(x) \\ \exists x P(x) & \equiv \neg\forall x \neg P(x) \end{array}$$

Osservazioni:

- I quantificatori aumentano notevolmente l'espressività delle formule del linguaggio della logica del prim'ordine, ma ne rendono anche più complessa la trattazione per mezzo di strumenti automatici di calcolo.
- Cerchiamo di trasformare le formule, mantenendone la semantica, in modo da ridurre i loro quantificatori.

**Definizione:** Una formula della logica dei predicati  $\varphi$  è in **forma normale prenessa** quando si ha che:

- $\varphi$  ha la forma  $Q_1 x_1 \dots Q_n x_n \psi$
- $\psi$  è una formula senza quantificatori
- $Q_i \in \{\forall, \exists\}$  con  $i = 1, \dots, n$

**Teorema:** Ogni formula della logica del prim'ordine può essere trasformata in una formula equivalente in forma normale prenessa.

Come procedere:

- Trasformazione in forma normale congiuntiva:
  - Eliminazione delle equivalenze  $\Leftrightarrow$
  - Eliminazione dei condizionali  $\Rightarrow$
  - Applicazione ripetuta delle leggi di DeMorgan e della legge distributiva
- Ridenominazione delle variabili se necessario
- Spostamento dei quantificatori universali all'inizio della formula
- **Skolemizzazione:** sostituzione di variabili esistenzialmente quantificate con nuove funzioni di Skolem
- Omissione dei quantificatori universali (con l'assunto che tutte le variabili presenti siano universalmente quantificate)

**Skolemizzazione:** è una tecnica per eliminare i quantificatori esistenziali. L'idea di base è questa. Data la formula:

“per ogni  $x$ , se  $x$  ha la proprietà  $P$ , allora esiste un  $y$  che è nella relazione  $Q$  con  $x$ .”

$$\forall x(P(x) \Rightarrow \exists y Q(x, y))$$

È come se la variabile  $y$  esistesse in funzione di  $x$ . Esprimiamo questo legame proprio con una funzione matematica (**la funzione di Skolem**), in cui  $y$  è funzione di  $x$ , e riscriviamo la formula così:

$$\forall x \left( P(x) \Rightarrow Q(x, f(x)) \right)$$

Attenzione: la nuova formula NON è equivalente a quella originale, perché facciamo una selezione di un singolo termine  $f(x)$  da mettere in relazione con  $x$

**Teorema di completezza di Gödel :** La logica del prim'ordine è **completa**, ossia, esiste un calcolo (insieme di regole di inferenza) con il quale ogni proposizione che è una conseguenza logica di una base di conoscenza può essere dimostrata.

$$\text{Se } KB \models \alpha, \text{ allora } KB \vdash \alpha$$

**Teorema (Correttezza) :** Esistono calcoli con i quali si possono dimostrare solo formule della logica del prim'ordine vere. Cioé:

$$\text{Se } KB \vdash \alpha, \text{ vale, allora } KB \models \alpha$$

**Eliminazione di  $\forall$**

$$\frac{\forall x P(x)}{P[x/t]}$$

**Eliminazione di  $\exists$**

$$\frac{\exists x Q(x)}{Q[x/K]}$$

**Introduzione di  $\exists$**

$$\frac{Q(K)}{\exists x Q(x)}$$

**NB:** con  $K$  nome di costante nuovo, non presente nella  $KB$ . Attenzione:  $\exists$  NON deve trovarsi nel campo di applicazione di un  $\forall$ . La nuova formula che ne deriva viene aggiunta alla  $KB$  affinché mi aiuti con il pattern matching all'interno di una dimostrazione meccanizzata.

## 3.2 Risoluzione nella logica del prim'ordine

Procediamo come nella logica predicativa, con negazione della tesi, trasformazione in clausole, e ricerca della clausola vuota.

*Dimostrazione di Child(Eve, Oscar, Anne):*

*Tesi :*

$$\text{Child}(Eve, Oscar, Anne)$$

*TN :*

$$\neg \text{Child}(Eve, Oscar, Anne)$$

*KB :*

$$\dots, \forall x \forall y \forall z \text{ Child}(x, y, z) \Rightarrow \text{Child}(x, z, y)$$

*KB'*

$$KB \cup \{TN\}$$

$$\forall x \forall y \forall z \text{ Child}(x, y, z) \Rightarrow \text{Child}(x, z, y) \equiv \neg \text{Child}(x, y, z) \vee \text{Child}(x, z, y)$$

*Come gestiamo i Predicati ?*

**Definizione:** Due letterali\*<sup>[1]</sup> sono chiamati **unificabili** se esiste una sostituzione  $\sigma$  per tutte le variabili presenti che rende uguali i letterali. Tale sostituzione  $\sigma$  è chiamata **unificatore**. Un unificatore è chiamato **unificatore più generale** (MGU: most general unifier) se tutti gli altri unificatori posso essere ottenuti a partire da esso tramite sostituzione di variabili.

*Dati i due letterali  $\text{Child}(x, y, z)$  e  $\text{Child}(Eve, Anne, Oscar)$ , esiste una sostituzione che li rende uguali:*

$$\sigma = [x/Eve, y/Anne, z/Oscar]$$

**Definizione:** La risoluzione tra due clausole in forma normale congiuntiva nella logica del prim'ordine si esegue come segue, dove  $\sigma$  è l'unificatore più generale di  $B$  e  $B'$

$$\frac{(A_1 \vee \dots \vee A_m \vee B), (\neg B' \vee C_1 \vee \dots \vee C_n) \quad \sigma(B) = \sigma(B')}{(\sigma(A_1) \vee \dots \vee \sigma(A_m) \vee \sigma(C_1) \vee \dots \vee \sigma(C_n))}$$

Attenzione: la sostituzione che unifica i due letterali va applicata anche agli altri che fanno parte delle clausole con cui si esegue la risoluzione.

*Continuando la dimostrazione precedente:*

$$\forall x \forall y \forall z \text{ Child}(x, y, z) \Rightarrow \text{Child}(x, z, y) \equiv \neg \text{Child}(x, y, z) \vee \text{Child}(x, z, y)$$

$\neg \text{Child}(\text{Eve}, \text{Oscar}, \text{Anne})$	[TN]
$\neg \text{Child}(x, y, z) \vee \text{Child}(x, z, y)$	[clausola 1 $\in KB$ ]
$\text{Child}(\text{Eve}, \text{Anne}, \text{Oscar})$	[clausola 2 $\in KB$ ]
$\text{Child}(\text{Eve}, \text{Oscar}, \text{Anne})$	[clausola 3, per risoluzione tra 1 e 2 con $\sigma = [x/\text{Eve}, y/\text{Anne}, z/\text{Oscar}]$ ]
{}	[risoluzione tra la TN e la clausola 3 c.v.d.]

**Teorema:** La regola della risoluzione è corretta: la clausola risultante è una conseguenza logica delle due clausole su cui la risoluzione viene eseguita. **Per la completezza serve un'ulteriore precisazione.**

**Il paradosso del barbiere:**

C'è un barbiere che rade tutti quelli che non si radono.

Ma il barbiere si rade o meno? facciamo finta che si rada, quindi questo barbiere rade se stesso, quindi lui non è tra quelli che non si radono, ma il barbiere rade quelli che non si radono e lui rade se stesso, quindi lui non si rade ma siamo partiti dicendo che si radesse

Il paradosso emerge quando ci si chiede se questo barbiere si rade o meno. Questa affermazione è contraddittoria, nel senso che è insoddisfacibile. Vogliamo dimostrarlo con la risoluzione.

*La frase tradotta in formula del prim'ordine diventa:*

Barbiere	costante
Rade	predicato, 1° chi rade, 2° il rasato
$\text{Rade}(x, x)$	rado me stesso

$$\forall x (\text{Rade}(\text{Barbiere}, x) \Leftrightarrow \neg \text{Rade}(x, x))$$

$$\begin{aligned} & \forall x \left( (\text{Rade}(\text{Barbiere}, x) \Rightarrow \neg \text{Rade}(x, x)) \wedge (\neg \text{Rade}(x, x) \Rightarrow \text{Rade}(\text{Barbiere}, x)) \right) \\ & \forall x \left( (\neg \text{Rade}(\text{Barbiere}, x) \vee \neg \text{Rade}(x, x)) \wedge (\text{Rade}(x, x) \vee \text{Rade}(\text{Barbiere}, x)) \right) \end{aligned}$$

trasformata in clausole, dà il seguente insieme di clausole, il quantificatore universale è sottointeso, ossia tutte le variabili sono quantificate:

$\neg \text{Rade}(\text{Barbiere}, x) \vee \neg \text{Rade}(x, x)$	[clausola 1]
$\text{Rade}(x, x) \vee \text{Rade}(\text{Barbiere}, x)$	[clausola 2]

Applicando la risoluzione viene eliminata una letterale per volta, dunque posso ottenere:

$\text{Rade}(x, x) \vee \neg \text{Rade}(x, x)$	[Risoluzione eliminazione $\text{Rade}(\text{Barbiere}, x)$ ]
$\text{Rade}(\text{Barbiere}, x) \vee \neg \text{Rade}(\text{Barbiere}, x)$	[Risoluzione eliminazione $\text{Rade}(x, x)$ ]

Sono delle tautologie  $\Rightarrow$  ci serve una **regola aggiuntiva**

**Definizione:** Si chiama **fattorizzazione** la seguente manipolazione sintattica di una clausola:

$$\frac{(A_1 \vee A_2 \vee \dots \vee A_n) \quad \sigma(A_1) = \sigma(A_2)}{\sigma(A_2) \vee \dots \vee \sigma(A_n)}$$

in cui se esistono due letterali che sono unificati da una sostituzione  $\sigma$ , allora possiamo derivare una clausola in cui applichiamo  $\sigma$  a tutti i letterali e un gruppo della coppia di letterali identici viene omesso  
 $\Rightarrow$  **eliminazione dei doppioni**

*Quindi, prima di procedere alla risoluzione, eseguiamo la fattorizzazione.*

$\neg Rade(Barbiere, x) \vee \neg Rade(x, x)$	[clausola 1]
$Rade(x, x) \vee Rade(Barbiere, x)$	[clausola 2]
$\neg Rade(Barbiere, Barbiere)$	[clausola 3, per fattorizzazione su 1, con $\sigma = [x/Barbiere]$ ]
$Rade(Barbiere, Barbiere)$	[clausola 4, per fattorizzazione su 2, con $\sigma = [x/Barbiere]$ ]
{}	[risoluzione tra 3 e 4 c.s.d]

**Teorema:** La regola di risoluzione insieme alla regola di fattorizzazione è completa per le **dimostrazioni per refutazione**. Cioè, applicando passaggi di fattorizzazione e risoluzione, la clausola vuota può essere derivata da qualsiasi insieme insoddisfacibile di formule in forma normale congiuntiva.

**Strategie di risoluzione:** Sebbene la completezza della risoluzione sia importante per l'utente, la ricerca di una prova può essere molto frustrante nella pratica. Anche se all'inizio ci sono solo pochissime coppie di clausole in  $KB' = KB \wedge \neg Q$  il dimostratore (umano o automatico) genera una nuova clausola ad ogni passo di risoluzione, il che aumenta il numero di possibili passi di risoluzione nell'iterazione successiva. Pertanto è stato a lungo tentato di ridurre lo spazio di ricerca utilizzando strategie speciali:

- **La risoluzione unitaria** dà la priorità ai passaggi di risoluzione in cui una delle due clausole è costituita da un solo letterale, chiamato **clausola unitaria**. Questa strategia preserva la completezza e porta in molti casi, ma non sempre, a una riduzione dello spazio di ricerca.  $\rightarrow$  **euristica completa**
- Applicazione **dell'insieme di supporto SOS** (sottoinsieme di  $KB \wedge \neg Q$ ), riduzione garantita dello spazio di ricerca. Ad ogni passo di risoluzione viene utilizzata una clausola in SOS e il risultato viene aggiunto ad SOS.  $\rightarrow$  **incompleta**
- Nella strategia della **risoluzione dell'input**, una clausola del set di input  $KB \wedge \neg Q$  deve essere coinvolta in ogni passo di risoluzione.  $\rightarrow$  **incompleta**
- **La regola del letterale puro** tutte le clausole che contengono letterali per i quali non ci sono letterali complementari in altre clausole possono essere eliminate.  $\rightarrow$  **completa**.
- Regola di **sussunzione** Se i letterali di una clausola  $C1$  rappresentano un sottoinsieme dei letterali della clausola  $C2$ , è possibile eliminare  $C2$   $\rightarrow$  **completa**.

\*[1] **Osservazione:** nella logica proposizionale, un letterale è una variabile proposizionale positiva o negata. Nella logica del primo ordine è una **formula atomica** o la negazione di una formula atomica. Una **formula atomica** nella logica del primo ordine è un predicato che prende argomenti costanti o variabili,

## 4 Risoluzione Logica

Abbiamo 2 modi per effettuare una dimostrazione:

- Semplice inferenza per arrivare alla Tesi
- Dimostrazione per contraddizione + Risoluzione

## 4.1 Dimostrazione per inferenza

Provare che  $\text{Child}(\text{Eve}, \text{Oscar}, \text{Anne})$  è conseguenza semantica di  $KB$ , ossia:  
 $KB \models \text{Child}(\text{Eve}, \text{Oscar}, \text{Anne})$

**Dimostrazione:**

- |    |   |  |
|----|---|--|
| 1) | $\text{Child}(\text{Eve}, \text{Anne}, \text{Oscar})$   | $[\text{Premessa}/KB]$   |
| 2) | $\forall x \forall y \forall z \text{Child}(x, y, z) \Rightarrow \text{Child}(x, z, y)$                               | $[\text{Premessa}/KB]$   |
| 3) | $\text{Child}(\text{Eve}, \text{Anne}, \text{Oscar}) \Rightarrow \text{Child}(\text{Eve}, \text{Oscar}, \text{Anne})$ | $[2 \text{ con } [x/\text{Eve}, y/\text{Anne}, z/\text{Oscar}]]$ |
| 4) | $\text{Child}(\text{Eve}, \text{Oscar}, \text{Anne})$   | $[\text{MP}(1,3)]$   |

Se le premesse sono entrambe vere allora per forza  
in output ottengo  $\text{Child}(\text{Eve}, \text{Oscar}, \text{Anne})$ :

$$MP(1,3) = \frac{\text{Child}(\text{Eve}, \text{Anne}, \text{Oscar}), \quad \text{Child}(\text{Eve}, \text{Anne}, \text{Oscar}) \Rightarrow \text{Child}(\text{Eve}, \text{Oscar}, \text{Anne})}{\text{Child}(\text{Eve}, \text{Oscar}, \text{Anne})}$$

## 4.2 Dimostrazione per contraddizione + Risoluzione

- Ricordarsi che le  $KB$  sono una congiunzione di clausole
- Le clausole sono una disgiunzione di letterali
- Per effettuare la risoluzione devo trasformare le  $KB$  in clausole
- Le nuove formule derivate vengono aggiunte alla  $KB$  per il matching
- Non devo applicare alcuna regola di inferenza solo risoluzione tra le clausole

**Dimostrazione di  $\text{Child}(\text{Eve}, \text{Oscar}, \text{Anne})$ :**

$$KB \Rightarrow \text{Child}(\text{Eve}, \text{Oscar}, \text{Anne}) \quad \text{ossia} \quad KB \wedge \neg \text{Child}(\text{Eve}, \text{Oscar}, \text{Anne})$$

Trasformo in clausole

$$\forall x \forall y \forall z \text{Child}(x, y, z) \Rightarrow \text{Child}(x, z, y) \equiv \neg \text{Child}(x, y, z) \vee \text{Child}(x, z, y)$$

<i>Tesi :</i>	$\text{Child}(\text{Eve}, \text{Oscar}, \text{Anne})$
<i>TN :</i>	$\neg \text{Child}(\text{Eve}, \text{Oscar}, \text{Anne})$
<i>KB'</i>	$KB \cup \{TN\}$

*Calcoli*

- |  |  |
|--|--|
| $\neg \text{Child}(\text{Eve}, \text{Oscar}, \text{Anne})$ | $[\text{TN}]$  |
| $\neg \text{Child}(x, y, z) \vee \text{Child}(x, z, y)$    | $[\text{clausola 1} \in KB]$   |
| $\text{Child}(\text{Eve}, \text{Anne}, \text{Oscar})$      | $[\text{clausola 2} \in KB]$   |
| $\text{Child}(\text{Eve}, \text{Oscar}, \text{Anne})$      | $[\text{clausola 3, per risoluzione tra 1 e 2 con } \sigma = [x/\text{Eve}, y/\text{Anne}, z/\text{Oscar}]]$ |
| $\{\}$   | $[\text{risoluzione tra la TN e la clausola 3 c.v.d.}]$  |

**Risoluzione clausola 3:** applico la unificazione generale

$$Ris = \frac{\text{clausola 1, clausola 2}}{\text{Clausola 3}}$$

$$Ris = \frac{\neg \text{Child}(\text{Eve}, \text{Anne}, \text{Oscar}) \vee \text{Child}(\text{Eve}, \text{Oscar}, \text{Anne}), \quad \text{Child}(\text{Eve}, \text{Anne}, \text{Oscar})}{\text{Child}(\text{Eve}, \text{Oscar}, \text{Anne})}$$

## 5 PROLOG

- Se si desidera implementare algoritmi, che inevitabilmente hanno componenti procedurali, una descrizione puramente dichiarativa è spesso insufficiente. Robert Kowalski, uno dei pionieri della programmazione logica, ha sottolineato questo punto con la formula:

$$\text{Algoritmo} = \text{Logica} + \text{Controllo} \rightarrow \text{da cui nasce PROLOG}$$

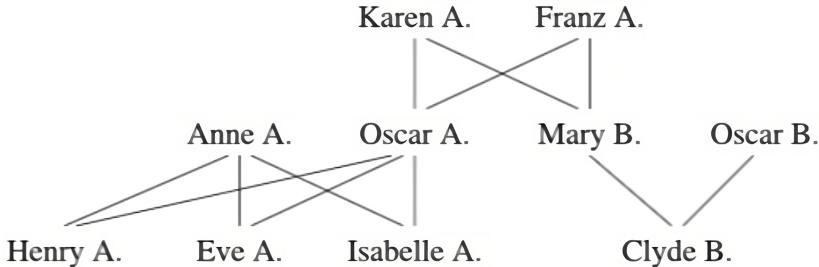
- La sintassi del linguaggio PROLOG consente solo clausole di Horn

### 5.1 Sintassi

- $A_1, \dots, A_m, A, B$  sono letterali
- I letterali, come in PL1 (logica del primo ordine), sono costituiti da simboli di predicato con termini come argomenti
- In PROLOG non ci sono negazioni in senso stretto logico perché il segno (positivo o negativo) di un letterale è determinato dalla sua posizione nella clausola.

PL1/clause normal form	PROLOG	Description
$(\neg A_1 \vee \dots \vee \neg A_m \vee B)$	$B :- A_1, \dots, A_m.$	Rule
$(A_1 \wedge \dots \wedge A_m) \Rightarrow B$	$B :- A_1, \dots, A_m.$	Rule
$A$	$A.$	Fact
$(\neg A_1 \vee \dots \vee \neg A_m)$	$? - A_1, \dots, A_m.$	Query
$\neg(A_1 \wedge \dots \wedge A_m)$	$? - A_1, \dots, A_m.$	Query

### 5.2 Esempio



$$\begin{aligned}
 KB \equiv & \text{female(karen)} \wedge \text{female(anne)} \wedge \text{female(mary)} \\
 & \wedge \text{female(eve)} \wedge \text{female(isabelle)} \\
 & \wedge \text{child(oscar, karen, franz)} \wedge \text{child(mary, karen, franz)} \\
 & \wedge \text{child(eve, anne, oscar)} \wedge \text{child(henry, anne, oscar)} \\
 & \wedge \text{child(isabelle, anne, oscar)} \wedge \text{child(clyde, mary, oscarb)} \\
 & \wedge (\forall x \forall y \forall z \text{child}(x, y, z) \Rightarrow \text{child}(x, z, y)) \\
 & \wedge (\forall x \forall y \text{descendant}(x, y) \Leftrightarrow \exists z \text{child}(x, y, z)) \\
 & \vee (\exists u \exists v \text{child}(x, u, v) \wedge \text{descendant}(u, y))).
 \end{aligned}$$

*Trascurando i predicati sul genere degli individui, ecco la KB sotto forma di programma PROLOG:*

- Le variabili → lettura maiuscola  $\{X, Y, Z, \dots\}$
- Le costanti → lettura minuscola  $\{oscar, karen, frank, \dots\}$
- I predicati → lettera minuscola  $\{child, descendant\}$
- Le **regole/rule** contengono le variabili
- I **fatti/fact** contengono costanti
- Le variabili che compaiono in grassetto sono le **variabili libere**
- Possiamo ammettere nella KB riformulata per PROLOG solo quelle clausole che sono di Horn (zero o al massimo un letterale positivo, tutti gli altri negati).
- Con PROLOG è possibile fare il condizionale solo in un verso: da dx verso sinistra, per cui  $\not\leftarrow \Leftarrow$
- PROLOG va in profondità finché non trova il risultato sulla prima regola con cui fa il match  $\Rightarrow$  problema di prolog si può finire in un loop infinito

```

child(oscar, karen, frank).
child(mary, karen, frank).
child(eve, anne, oscar).
child(henry, anne, oscar).
child(isolde, anne, oscar).
child(clyde, mary, oscarb).

child(X,Z,Y) :- child(X, Y, Z).

descendant(X,Y) :- child(X,Y,Z).
descendant(X,Y) :- child(X,U,V), descendant(U,Y).

```

### 5.3 Funzionamento di PROLOG

- Io do una query a PROLOG per vedere se è vera
- L'interprete di PROLOG cerca il matching all'interno della *KB* partendo dall'alto e scendendo verso il basso
- Appena c'è un match, si va a vedere quello che c'è a dx
- Se incontro una **regola** quest'ultima viene applicata alla query (ossia quello che c'è a dx della regola viene applicata alla query per crearne una nuova) e si riparte dall'inizio ripercorrendo tutto per vedere se c'è un match sulla nuova query. Se una delle alternative fallisce, ciò si traduce in un backtracking all'ultimo punto di diramazione e viene testata l'alternativa successiva.
- Occorre prestare attenzione a non finire dentro loop infiniti.
- Se il risultato della query è **true** vuol dire che la query è effettivamente una conseguenza logica della *KB* e l'interprete riesce a lavorare con la clausola di Horn per dimostrarne la verità.
- L'interprete se da un avvertimento di “**singleton variable**” questo può rappresentare: un errore di ortografia oppure dimenticare di associare o usare una variabile. Semplicemente l'interprete vuole accertarsi che queste variabili non utilizzate non siano un errore di ortografia oppure sono poi usate in altre clausole che l'utente si è dimenticato di inserire.
- **L'unificazione** viene utilizzata per cercare una corrispondenza tra la **nuova** query e i fatti presenti nei dati di input. L'obiettivo è trovare un fatto che sia unificabile con la query, cioè un fatto che abbia lo stesso predicato nella testa della clausola.

## 5.4 Backtracking

L'interprete di Prolog esplora le possibili soluzioni, potrebbe dover tornare indietro e provare altre possibilità per trovare tutte le soluzioni valide. Il backtracking è simile a un processo di esplorazione di un albero decisionale, in cui il motore prova un percorso, esplora le sue sotto-opzioni e, se non trova una soluzione valida, torna al punto di decisione precedente e prova una nuova opzione.

```
?- trace, q(X).  
  
% Risultato  
Call:q(_4154)  
  Call:p(_428)  
    Exit:p(a)  
    Exit:q(a)  
    X = a  
  
p(a).  
p(b).  
p(c).  
  
q(X) :- p(X).  
q(d).  
  
Redo:p(_428)  
  Exit:p(b)  
  Exit:q(b)  
  X = b  
  
Redo:p(_422)  
  Exit:p(c)  
  Exit:q(c)  
  X = c  
  
Redo:q(_422)  
  Exit:q(d)  
  X = d
```

In questo esempio, la query `?- q(X).` cercherà di trovare tutte le soluzioni per il predicato `q(X)`. Prolog inizierà esplorando il primo cammino possibile, che è soddisfare `q(X)` unificando `X` con `a`, e restituirà `X = a`.

Dopodiché, Prolog tornerà indietro al punto di decisione e proverà un altro cammino. Continuerà così fino a quando tutte le soluzioni possibili saranno state trovate e visualizzate.

## 5.5 Risoluzione query:

- ?- child(eve, oscar, anne).

Si prova a fare il matching con i primi 6 fatti senza avere un riscontro. L'interprete Prolog incontra la regola ricorsiva e **unificata** la query con tale regola nella riga 8. Si riparte dalla riga 1 dove ora il sistema tenta di risolvere la nuova query `child(eve, anne, oscar)` che fa il match con il 3 fatto.

```

child(oscar, karen, frank).
child(mary, karen, frank).
child(eve, anne, oscar).
child(henry, anne, oscar).
child(isolde, anne, oscar).
child(clyde, mary, oscarb).

child(X,Z,Y) :- child(X,Y,Z).
descendant(X,Y) :- child(X,Y,Z).
descendant(X,Y) :- child(X,U,V), descendant(U,Y).

?- trace, child(eve, oscar, anne).

```

*% Risultato:*  
**Call:child(eve,oscar,anne)**  
**Call:child(eve,anne,oscar)**  
**Exit:child(eve,anne,oscar)**  
**Exit:child(eve,oscar,anne)**  
**true**

- ?- descendant(clyde, karen).

Loop infinito. La ragione di ciò è che dopo aver eseguito la riga 10 la nuova query si richiama ricorsivamente senza possibilità di risoluzione alla riga 8. Per eliminare il problema si introduce

```

?- trace, descendant(clyde, karen).

% Risultato:
Call:descendant(clyde,karen)
Call:child(clyde,karen,_808)
Fail:child(clyde,karen,_808)
Redo:descendant(clyde,karen)
Call:child(clyde,_806,karen)
Fail:child(clyde,_806,karen)
Redo:descendant(clyde,karen)
Call:child(clyde,_678,_810)
Exit:child(clyde,mary,oscarb)
Call:descendant(mary,karen)
Call:child(mary,karen,_812)
Exit:child(mary,karen,frank)
Exit:descendant(mary,karen)
Exit:descendant(clyde,karen)
true

```

- ?- child(eve,oscar,anne).

riceve risposta **false** perché la simmetria del predicato `child` nelle ultime due variabili non è più data. Introducendo il nuovo predicato `child_fact` per i fatti, il predicato `child` non è più ricorsivo.

```

child_fact(oscar, karen, frank).
child_fact(mary, karen, frank).
child_fact(eve, anne, oscar).
child_fact(henry, anne, oscar).
child_fact(clyde, mary, oscarb).
child(X,Z,Y) :- child_fact(X,Y,Z).
child(X,Z,Y) :- child_fact(X,Z,Y).
descendant(X,Y) :- child(X,Y,Z).
descendant(X,Y) :- child(X,U,V), descendant(U,Y).

?- trace, child(eve,oscar,anne).

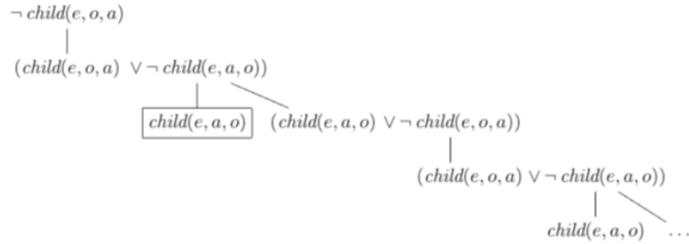
% Risultato:
Call:child(eve,oscar,anne)
Call:child_fact(eve,anne,oscar)
Exit:child_fact(eve,anne,oscar)
Exit:child(eve,oscar,anne)
true

```

## 5.6 Distinzione tra semantica dichiarativa e procedurale in PROLOG

- La semantica dichiarativa è data dall'interpretazione logica delle clausole di Horn, dunque definisce il significato delle clausole e fornisce un'interpretazione logica dei predici definiti nel programma.
- La semantica procedurale, al contrario, è definita dall'esecuzione del programma PROLOG. Essa riguarda l'ordine di valutazione delle clausole e la strategia di risoluzione utilizzata per trovare soluzioni alle query
- La risoluzione di: `child(eve, oscar, anne)` è rappresentata come un albero di ricerca.

L'esecuzione inizia in alto a sinistra con la query. Ciascun segmento rappresenta un possibile passo di risoluzione SLD (“Selection-ruledriven Linear resolution for Definite clauses”, risoluzione lineare basata su regole di selezione per clausole definite) con un letterale complementare unificabile. Mentre l'albero di ricerca diventa infinitamente profondo dalla regola ricorsiva, l'esecuzione di PROLOG termina perché i fatti si verificano prima della regola nei dati di input.



## 5.7 Elementi procedurali e controllo dell'esecuzione

- È importante controllare l'esecuzione di PROLOG
- Evitare in particolare il backtracking non necessario può portare a grandi aumenti di inefficienza.
- A tal fine esiste lo strumento del “taglio”. Inserendo un punto esclamativo in una clausola, possiamo impedire il backtracking in quel punto. Essenzialmente rappresenta la prima soluzione che trova senza esplorare più nulla
- Nel seguente programma, il predicato `max (X, Y, Max)` calcola il massimo dei due numeri X e Y.

```

?- trace, max(9,10,Z).

% Risultato:
Call:max(9,10,_4392)
Call:9>=10
Fail:9>=10
Redo:max(9,10,_434)
Call:9<10
Exit:9<10
Exit:max(9,10,10)
Z = 10
  
```

- Se si applica il primo caso (prima clausola), il secondo non verrà raggiunto.
- D'altra parte, se il primo caso non si applica, la condizione del secondo caso è (necessariamente) vera (perché i due casi sono mutuamente esclusivi ed esaustivi), il che significa che non è necessario verificarlo.
- Possiamo ottimizzare il tutto con un taglio: la seconda clausola viene chiamata solo se è veramente necessario, cioè se la prima clausola fallisce

```

?- trace, max(9,10,Z).

% Risultato:
Call:max(9,10,_5382)
Call:9>=10
Fail:9>=10
Redo:max(9,10,_432)
Exit:max(9,10,10)
Z = 10
  
```

$$max(X, Y, X) : - X >= Y. \quad \text{if } \text{condizione}$$

## 5.8 Liste

- Una lista con gli elementi a,2,2,b,3,4,5 ha la forma: [a,2,2,b,3,4,5]
- Il costrutto [H|T] separa il primo elemento H ossia la testa/head, dal resto T tail/coda.

```
list([a, 2, 2, b, 3, 4, 5]).  
?- list([H|T]).  
  
risultato:  
H = a,  
T = [2, 2, b, 3, 4, 5]
```

- Esempio:

```
append([],L,L).  
append([X|L1], L2, [X|L3]) :- append(L1,L2,L3).  
?- trace, append([a, b, c], [d, 1, 2], Z).  
  
risultato:  
Call:append([a, b, c], [d, 1, 2], _4306)  
Call:append([b, c], [d, 1, 2], _692)  
Call:append([c], [d, 1, 2], _698)  
Call:append([], [d, 1, 2], _704)  
Exit:append([], [d, 1, 2], [d, 1, 2])  
Exit:append([c], [d, 1, 2], [c, d, 1, 2])  
Exit:append([b, c], [d, 1, 2], [b, c, d, 1, 2])  
Exit:append([a, b, c], [d, 1, 2], [a, b, c, d, 1, 2])  
Z = [a, b, c, d, 1, 2]
```

- Esempio:

```
append([],L,L).  
append([X|L1], L2, [X|L3]) :- append(L1,L2,L3).  
?- trace, append(X, [1,2,3], [4,5,6,1,2,3]).  
  
Risultato:  
Call:append(_4324, [1, 2, 3], [4, 5, 6, 1, 2, 3])  
Call:append(_710, [1, 2, 3], [5, 6, 1, 2, 3])  
Call:append(_716, [1, 2, 3], [6, 1, 2, 3])  
Call:append(_722, [1, 2, 3], [1, 2, 3])  
Exit:append([], [1, 2, 3], [1, 2, 3])  
Exit:append([6], [1, 2, 3], [6, 1, 2, 3])  
Exit:append([5, 6], [1, 2, 3], [5, 6, 1, 2, 3])  
Exit:append([4, 5, 6], [1, 2, 3], [4, 5, 6, 1, 2, 3])  
X = [4, 5, 6]
```

- Esempio:

```
accrev([], A, A).  
accrev([H|T], A, R) :- accredv(T, [H|A], R).  
?- trace, accredv([1,2,3], [], Z).  
  
Ossia:  
accrev([lista da invertire], [accumulatore], [lista invertita])  
  
Risultato:  
Call:accrev([1, 2, 3], [], _4298)  
Call:accrev([2, 3], [1], _456)  
Call:accrev([3], [2, 1], _456)  
Call:accrev([], [3, 2, 1], _456)  
Exit:accrev([], [3, 2, 1], [3, 2, 1])  
Exit:accrev([3], [2, 1], [3, 2, 1])  
Exit:accrev([2, 3], [1], [3, 2, 1])  
Exit:accrev([1, 2, 3], [], [3, 2, 1])  
Z = [3, 2, 1]
```

## 6 Search

- Supponiamo che il fattore di ramificazione sia una costante uguale a 30 e che la prima soluzione sia a profondità 50. L'albero di ricerca ha  $30^{50} \approx 7,2 \cdot 10^{73}$  nodi foglia, ma il numero totale di passaggi di inferenza è ancora maggiore, perché dobbiamo considerare non solo ogni nodo foglia ma anche ogni nodo interno dell'albero. Pertanto dobbiamo sommare i nodi su tutti i livelli e ottenere il numero totale di nodi dell'albero di ricerca:

$$\sum_{d=0}^{50} 30^d = \frac{1 - 30^{51}}{1 - 30} = 7,4 \times 10^{73}$$

Supponiamo di avere 10.000 computer che possono eseguire ciascuno un  $10^9$  di inferenze al secondo e supponiamo di poter distribuire il lavoro su tutti i computer senza alcun costo (overhead). Il tempo di calcolo totale per tutte le inferenze sarebbe approssimativamente uguale a:

$$\frac{7,4 \times 10^{73} \text{ inferences}}{10.000 \times 10^9 \text{ inferences/sec}} \approx 2.3 \times 10^{53} \text{ years}$$

Impossibile esplorare tutto lo spazio di ricerca in questo modo → adotiamo **strategie per la riduzione dello spazio di ricerca**.

- Esistono processi con i quali i computer possono, analogamente agli esseri umani, migliorare le loro strategie di ricerca mediante conoscenze pregresse programmate o anche l'apprendimento.

### 6.1 Definizioni

Abbiamo a che fare con problemi di ricerca discreti:

- essendo nello stato  $s$ , un'azione  $a_1$  porta a un nuovo stato  $s'$   $\Rightarrow s' = a_1(s)$
- un'azione diversa può portare allo stato  $s''$   $\Rightarrow s'' = a_2(s)$

L'applicazione ricorsiva di tutte le azioni possibili a tutti gli stati, a partire dallo stato iniziale, produce l'albero di ricerca.

**Definizioni:** un **problema di ricerca** è definito dai seguenti valori:

- **Stato:** descrizione dello stato del mondo in cui si trova l'agente di ricerca (essere umano o artificiale)
- **Stato iniziale:** lo stato iniziale in cui viene avviato l'agente di ricerca.
- **Stato obiettivo:** se l'agente raggiunge uno stato obiettivo, termina e genera una soluzione (se lo si desidera).
- **Azioni:** il complesso delle mosse che l'agente può compiere.
- **Soluzione:** il percorso nella struttura di ricerca dallo stato iniziale allo stato obiettivo.
- **Funzione costo:** assegna un valore di costo a ogni azione. Tale funzione è necessaria se si vuole trovare una soluzione ottimale in termini di costi.
- **Spazio degli stati:** insieme di tutti gli stati.
- **Fattore di ramificazione (branching factor):** il numero di stati successori (nodi) di uno stato  $s$ , indicato  $b(s)$  o  $b$  se il fattore di ramificazione è costante.
- **Fattore di ramificazione effettivo:** dato un albero di profondità  $d$  con  $n$  nodi totali, il suo fattore di ramificazione effettivo è definito come il fattore di ramificazione che avrebbe un albero con fattore di ramificazione costante, uguale profondità e uguale numero di nodi.

**Definizione:** un algoritmo di ricerca viene definito **completo** se trova una soluzione per ogni problema risolvibile. Se un algoritmo di ricerca completo termina senza trovare una soluzione, allora il problema è **irrisolvibile**.

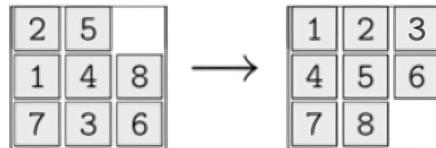
**Definizione:** Un algoritmo di ricerca è detto **ottimale** se trova sempre la soluzione con il costo più basso. (se esiste una soluzione)

**Definizione:** un problema viene definito **deterministico** quando ogni azione porta da uno stato a uno stato successore unico, viene invece definito **osservabile** quando l'agente sa sempre in quale stato si trova.

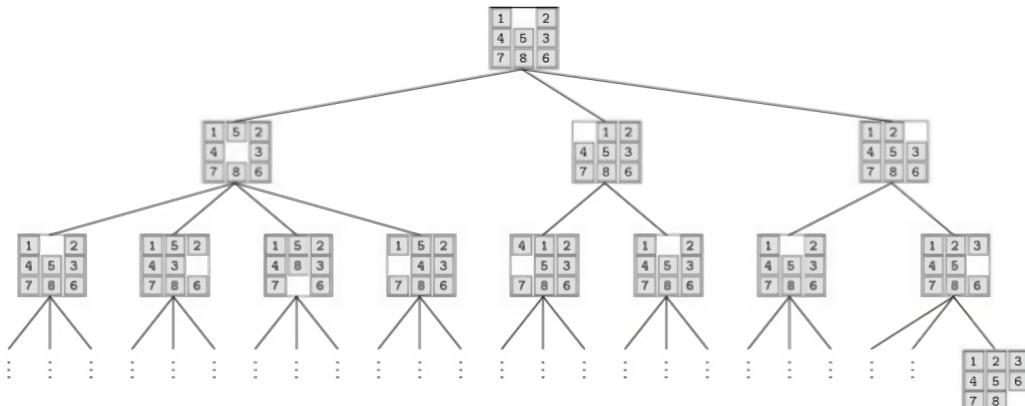
**Definizione:** Problemi deterministici e osservabili rendono la pianificazione delle azioni relativamente semplice perché, grazie a un modello astratto, è possibile trovare sequenze di azioni per la soluzione del problema senza eseguire effettivamente le azioni nel mondo reale → **algoritmi offline**. Gli **algoritmi online** prendono decisioni in base ai segnali dei sensori in ogni situazione.

## 6.2 Esempio

dato un puzzle rappresentato come una matrice  $3 \times 3$ , l'obiettivo è quello di ordinare i quadratini in ordine crescente per riga

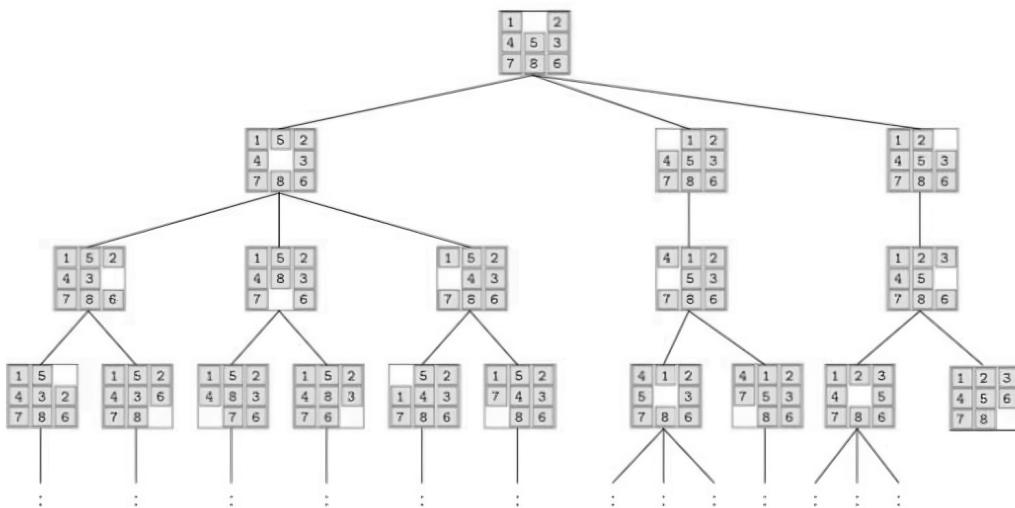


L'albero di ricerca per uno stato iniziale è rappresentato in figura (il goal è il nodo in fondo a destra):



**NB:** lo stato iniziale viene ripetuto più volte su due livelli più in basso di profondità perché in una semplice **uninformed search**, ogni azione può essere invertita nel passaggio successivo.

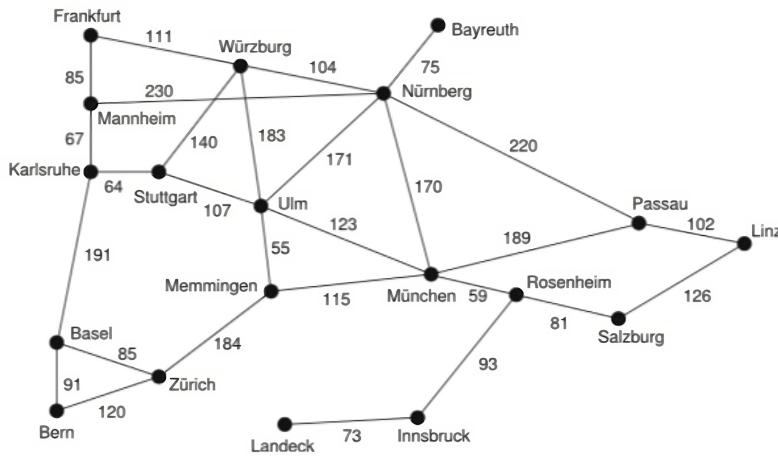
Se non permettiamo cicli di lunghezza 2 (in cui si ritorna a uno stato precedente annullando gli effetti della mossa appena fatta), allora per lo stesso stato iniziale otteniamo un albero di ricerca fatto come segue:



### 6.3 Esempio 2

Stiamo cercando un percorso ottimale dalla città A alla città B. Formalizzando il problema come un problema di ricerca abbiamo:

- **Stato:** una città come posizione corrente del viaggiatore.
- **Stato di partenza:** una città arbitraria.
- **Stato obiettivo:** una città arbitraria.
- **Azioni:** viaggiare dalla città corrente a una città vicina.
- **Funzione di costo:** la distanza tra le città.
- **Spazio degli stati:** tutte le città, ovvero i nodi del grafo.



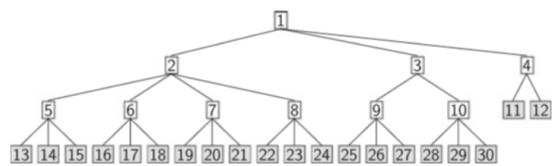
### 6.4 Uninformed search

- La ricerca senza informazione (uninformed search) si basa sul provare ciecamente tutte le possibilità
- Lo stato iniziale si può ripetere ai livelli inferiori di profondità perché in una semplice uninformed search, ogni azione può essere invertita nel passaggio successivo.
- Noi vedremo:
  - **Breadth-first search:** ricerca in ampiezza
  - **Depth-first search:** ricerca in profondità
  - **Uniform Cost Search:** ricerca in ampiezza con ordinamento di espansione
  - **Iterative deepening:** Depth search ma con limite di profondità iterato

### 6.5 Breadth-first search

```
BREADTHFIRSTSEARCH(NodeList, Goal)
NewNodes = ∅
For all Node ∈ NodeList
  If GoalReached(Node, Goal)
    Return("Solution found", Node)
    NewNodes = Append(NewNodes, Successors(Node))
  If NewNodes ≠ ∅
    Return(BREADTH-FIRST-SEARCH(NewNodes, Goal))
  Else
    Return("No solution")
```

*Istantanea della ricerca in ampiezza:  
con relativo ordine di percorrenza*



- Nella **ricerca in ampiezza**, l'albero di ricerca viene esplorato dall'alto verso il basso
- Prima ogni nodo nell'elenco dei nodi (NodeList) viene testato per verificare se si tratta di un nodo obiettivo e, in caso di successo, il programma viene interrotto. Altrimenti vengono generati tutti i successori del nodo

- La ricerca viene quindi proseguita in modo ricorsivo nell'elenco di tutti i nodi appena generati. Il tutto si ripete fino a quando non vengono generati più successori.
- **GoalReached** calcola se l'argomento è un nodo obiettivo e **Successors** calcola l'elenco di tutti i nodi successori del suo argomento
- la ricerca in ampiezza fa una ricerca completa di ogni profondità e raggiunge ogni profondità in un tempo finito, è **completa** se il fattore di ramificazione  $b$  è finito.
- La soluzione **ottimale** (cioè la più breve) si trova se i costi di tutte le azioni sono gli stessi.
- Il tempo di calcolo e lo spazio di memoria necessari crescono in modo esponenziale con la profondità dell'albero. Per un albero con fattore di ramificazione costante  $b$  e profondità  $d$ , **il tempo di calcolo totale è dato da:**

$$c \cdot \sum_{i=0}^d b^i = \frac{b^{d+1} - 1}{b - 1} = O(b^d)$$

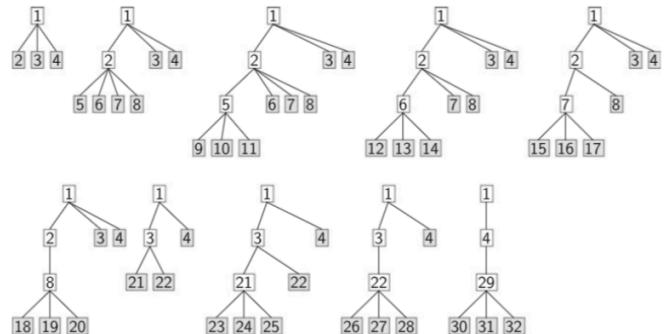
Sebbene solo l'ultimo livello venga salvato in memoria, anche lo **spazio di memoria richiesto** è  $O(b^d)$

- Il problema della soluzione più breve, non sempre trovata, può essere risolto con la cosiddetta **Uniform Cost Search**, in cui viene sempre espanso il nodo con il costo più basso dalla lista dei nodi (che viene ordinata in ordine crescente per costo), e i nuovi nodi vengono ordinati per costo. Così troviamo la soluzione ottimale.

**NB:** Il problema della memoria che si riempie non è stato ancora risolto. Una soluzione a questo problema è fornita dalla ricerca in profondità

## 6.6 Depth-first search

```
DEPTHFIRSTSEARCH(Node, Goal)
If GoalReached(Node, Goal) Return("Solution found")
NewNodes = Successors(Node)
While NewNodes ≠ ∅
    Result = DEPTH-FIRST-SEARCH(First(NewNodes), Goal)
    If Result = "Solution found" Return("Solution found")
    NewNodes = Rest(NewNodes)
Return("No solution")
```



- Vengono salvati in memoria solo pochi nodi alla volta
- Dopo l'espansione di un nodo vengono salvati solo i suoi successori e il primo nodo successore viene immediatamente espanso → così la ricerca diventa rapidamente molto profonda
- Solo quando un nodo non ha successori e la ricerca non riesce a quella profondità, il successivo nodo aperto viene espanso tramite il **backtracking** fino all'ultimo ramo e così via.
- In questa esecuzione della ricerca in profondità tutti i nodi a profondità 3 non sono obiettivo e causano il backtracking. I nodi sono numerati nell'ordine in cui sono stati generati.
- La ricerca in profondità richiede molta meno **memoria** rispetto alla ricerca in ampiezza perché al massimo  $b$  nodi vengono salvati a ciascuna profondità. Quindi abbiamo bisogno al massimo di  $b \cdot d$  celle di memoria.
- La ricerca in profondità **non è completa** per alberi infinitamente profondi perché la ricerca in profondità dà luogo a un ciclo infinito. A causa del ciclo infinito, non è possibile fornire alcun limite al tempo di calcolo
- Nel caso di un albero di ricerca finitamente profondo con profondità  $d$ , vengono generati un totale di circa  $b \cdot d$  nodi. Così il **tempo di calcolo** cresce, proprio come nella ricerca in ampiezza, esponenzialmente con la profondità.
- Possiamo rendere finito l'albero di ricerca impostando un limite di profondità. Ora, se non viene trovata alcuna soluzione nell'albero di ricerca ridotto, possono comunque esserci soluzioni al di là del limite impostato. Così la ricerca diventa incompleta

## 6.7 Approfondimento iterativo (iterative deepening)

```

ITERATIVEDEEPENING(Node, Goal)
DepthLimit = 0
Repeat
    Result = DEPTHFIRSTSEARCH-B(Node, Goal, 0, DepthLimit)
    DepthLimit = DepthLimit + 1
Until Result = "Solution found"

```

```

DEPTHFIRSTSEARCH-B(Node, Goal, Depth, Limit)
If GoalReached(Node, Goal) Return("Solution found")
NewNodes = Successors(Node)
While NewNodes ≠ ∅ And Depth < Limit
    Result =
        DEPTHFIRSTSEARCH-B(First(NewNodes), Goal, Depth + 1, Limit)
    If Result = "Solution found" Return("Solution found")
    NewNodes = Rest(NewNodes)
Return("No solution")

```

- Iniziamo la ricerca in profondità con un limite di profondità di 1. Se non viene trovata alcuna soluzione, aumentiamo il limite di 1 e iniziamo la ricerca dall'inizio, e così via.
- Il requisito di memoria è lo stesso della ricerca in profondità. Si potrebbe sostenere che il riavvio ripetuto della ricerca in profondità falla profondità zero causa molto lavoro ridondante.
- Si può dimostrare che la somma del numero di nodi di tutte le profondità fino a quella prima dell'ultima ( $d_{max} - 1$ ) in tutti i sottoalberi precedentemente esplorati è molto inferiore al numero di nodi nell'ultimo albero esplorato
- Proprio come la ricerca in ampiezza, questo metodo è completo e, dato un costo costante per tutte le azioni, trova la soluzione più breve.

### Confronto algoritmi

(\*) significa che l'affermazione è vera solo dato un costo di azione costante.  
 $d$  è la profondità massima per un albero di ricerca finito.

	Breadth-first search	Uniform cost search	Depth-first search	Iterative deepening
Completeness	Yes	Yes	No	Yes
Optimal solution	Yes (*)	Yes	No	Yes (*)
Computation time	$b^d$	$b^d$	$\infty$ or $b^d$	$b^d$
Memory use	$b^d$	$b^d$	$bd$	$bd$

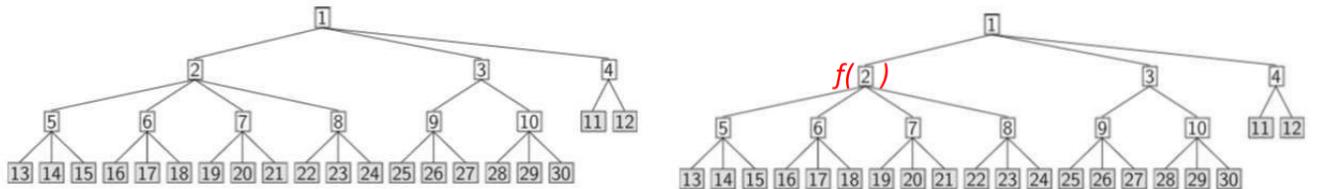
## 6.8 Controllo di cicli (cycle check)

- Come nell'esempio dell'8-puzzle, alcuni nodi possono essere visitati ripetutamente durante una ricerca
- Ogni mossa può essere annullata immediatamente, il che porta a cicli non necessari di lunghezza 2.
- Tali cicli possono essere prevenuti registrando all'interno di ogni nodo tutti i suoi predecessori e, quando si espande un nodo, confrontando i nodi successori appena creati con i nodi predecessori.
- Tutti i duplicati trovati possono essere rimossi dall'elenco dei nodi successori
- Questo controllo costa solo un piccolo fattore costante di spazio di memoria aggiuntivo e aumenta il tempo di calcolo costante  $c$  di un'ulteriore costante  $\delta$  per il controllo stesso, per un totale di  $c + \delta$
- Questo carico in memoria (si spera) viene compensato da una riduzione del costo della ricerca

## 7 Algoritmi di ricerca

- Finora abbiamo visto **uninformed search**, ossia algoritmi di ricerca che non fanno alcuna presupposizione sui nodi dell'albero e sui loro contenuti.
- **Euristica:** conoscenza acquisita tramite l'esperienza che ci può aiutare a risolvere problemi in maniera rapida, anche se il successo non è garantito.
- Otteniamo un algoritmo di **heuristic search** quando aggiungiamo alle istruzioni di un algoritmo di **uninformed search** una funzione di valutazione dello stato corrente e dei possibili stati successivi  $f(s)$  su cui basare le mosse successive, con lo scopo di risolvere il problema in maniera più rapida.

### 7.1 Heuristic breadth-first



- Nella versione **uninformed**, i nodi aperti sono espansi da sinistra a destra.
- Nella versione **heuristic**, invece, i nodi sono espansi in base alla loro **valutazione euristica**. Dall'insieme di nodi aperti, il nodo con la valutazione minima (cioè con la maggiore probabilità di portare alla soluzione con costi minimi) viene espanso per primo.
- Ciò si ottiene valutando immediatamente i nodi man mano che vengono espansi e ordinandoli nell'elenco dei nodi aperti.
- La versione **heuristic** della funzione **Successors**, che genera i successori (figli) di un nodo, deve anche calcolare immediatamente per questi nodi successori le loro valutazioni euristiche per permetterne l'ordinamento.

#### Algoritmo di Heuristic Search

```

HEURISTICSEARCH(Start, Goal)
NodeList = [Start]
While True
    If NodeList =  $\emptyset$  Return("No solution")
    Node = First(NodeList)
    NodeList = Rest(NodeList)
    If GoalReached(Node, Goal) Return("Solution found", Node)
    NodeList = SortIn(Successors(Node), NodeList)

```

**SortIn(X, Y)** inserisce gli elementi dall'elenco non ordinato X nell'elenco ordinato in ordine crescente Y. La valutazione euristica viene utilizzata come chiave di ordinamento. In questo modo è garantito che il nodo migliore (cioè quello con il valore euristico più basso) sia sempre all'inizio dell'elenco

Quando si fa l'inserimento ordinato di un nuovo nodo nell'elenco dei nodi, può essere vantaggioso verificare se il nodo è già presente nell'elenco e, in tal caso, eliminare il duplicato.

### 7.2 Euristica e stime

- La migliore euristica sarebbe una funzione che calcola i costi effettivi da ciascun nodo all'obiettivo. Fare ciò, tuttavia, richiederebbe un attraversamento dell'intero spazio di ricerca, che è esattamente ciò che l'euristica dovrebbe impedire.
- Un'idea interessante per trovare un'euristica è la semplificazione del problema

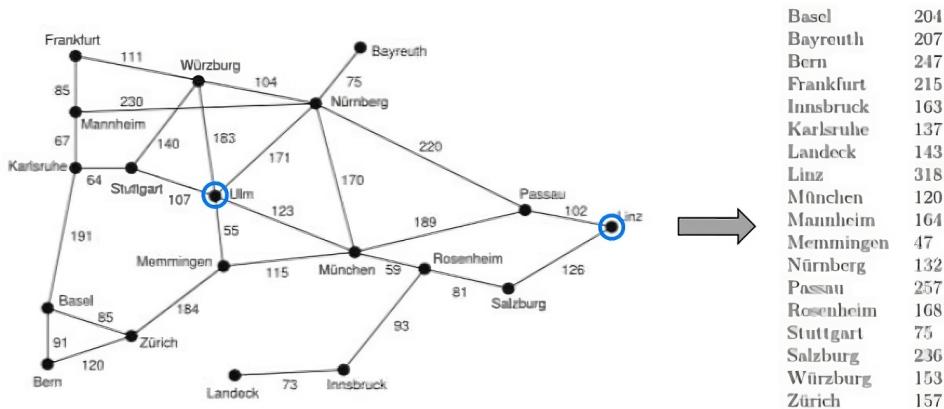
- Nella nuova versione, il calcolo dei costi è sufficientemente semplificato da poter essere risolto con un consumo limitato di risorse. Chiamiamo questa funzione di stima dei costi  $h(s)$ .
- **NB:** dunque  $f(s) = h(s)$ , si approssima la funzione di valutazione con la stima dei costi nel problema semplificato. Per la ricerca invece, si sceglie lo stato con il valore  $h$  stimato più basso dall'elenco degli stati attualmente disponibili.

### 7.3 Problema semplificato: esempio

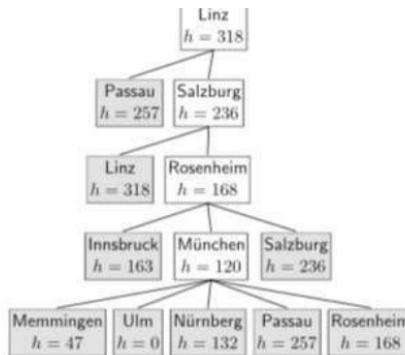
**Semplificazione problema:** invece di cercare il percorso ottimale, determiniamo prima da ogni nodo un percorso di volo verso l'obiettivo → Ulm.

**La funzione di costo semplificata:**

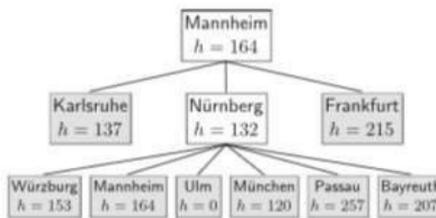
$$h(s) = \text{distanza di volo da } s \text{ a Ulm}$$



Con Ulm come obiettivo, scegliamo Linz come punto di partenza. Controlliamo a quali città Linz è connessa, e scegliamo quella con la distanza di volo a Ulm inferiore.



Purtroppo, questa ricerca non trova sempre la soluzione ottimale: es partendo da Mannheim



$$\text{Mannheim} - \text{Nürnberg} - \text{Ulm} = 401\text{km}$$

$$\text{Mannheim} - \text{Karlsruhe} - \text{Stuttgart} - \text{Ulm} = 238\text{km}$$

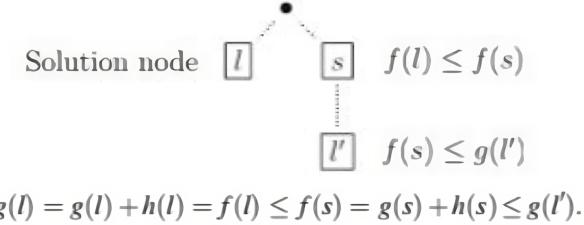
**NB:** l'euristica guarda avanti **avidamente** solo all'obiettivo invece di tener conto anche del tratto che è già stato tracciato fino al nodo attuale. Per questo motivo, questa ricerca si chiama **greedy**.

## 7.4 Algoritmo A\*

- Euristica guarda solo all'obiettivo senza tenere conto di altro → **ricerca greedy**
- Prendendo in considerazione i **costi accumulati** durante la ricerca fino al nodo attuale  $s$ , definiamo la funzione di costo:  $g(s) = \text{somma dei costi maturati dall'inizio fino al nodo } s$ .

$$\begin{aligned}
 \text{Funzione di valutazione:} & f(s) \\
 \text{Funzione costi semplificata/euristica:} & h(s) \\
 \text{Funzione di costi maturati:} & g(s) = g(s - 1) + g_{\text{passo}} \\
 \text{Nuova funzione di valutazione:} & f(s) = h(s) + g(s)
 \end{aligned}$$

- **Definizione:** una funzione di stima euristica dei costi  $h(s)$  che non sovrastima mai il costo effettivo del percorso dallo stato  $s$  fino all'obiettivo è definita **ammissibile**.
- La funzione `HeuristicSearch` insieme a una funzione di valutazione  $f(s) = g(s) + h(s)$  con una funzione euristica ammissibile  $h$  è chiamata **algoritmo A\*** (letto **A star**). Questo algoritmo è **completo e ottimale**, ossia A\* trova sempre la soluzione più breve per ogni problema di ricerca risolvibile.
- L'idea dell'algoritmo A\* è di selezionare i nodi da esplorare in base al valore di  $f(s)$  - ovvero il costo accumulato finora  $g(s)$  + la stima euristica del costo rimanente  $h(s)$ . Questo garantisce che l'algoritmo esplori inizialmente i nodi che sembrano promettenti in termini di minimizzare il costo totale della soluzione.
- Nell'algoritmo `HeuristicSearch`, ogni nodo  $s$  appena generato viene ordinato in base alla funzione `SortIn` in base alla sua **valutazione euristica  $f(s)$** . Il nodo con il valore di valutazione più piccolo si trova quindi all'inizio della lista. Se il nodo  $l$  all'inizio dell'elenco è un nodo di soluzione, nessun altro nodo ha una valutazione euristica migliore. Per tutti gli altri nodi  $s$  è vero allora che  $f(l) \leq f(s)$ . Poiché l'euristica è **ammissibile** (non ci sono sovrastime di costi), non è possibile trovare una soluzione migliore  $l'$ , anche dopo l'espansione di tutti gli altri nodi.



- La prima uguaglianza vale perché  $l$  è un nodo soluzione con  $h(l) = 0$
- La seconda è la definizione di  $f \rightarrow f(l) = g(l) + h(l)$
- La terza (dis)eguaglianza vale perché l'elenco dei nodi aperti è ordinato in ordine crescente.
- La quarta uguaglianza è ancora una volta la definizione di  $f$ .
- L'ultima (dis)eguaglianza è l'ammissibilità dell'euristica, che non sovrastima mai il costo del percorso dal nodo  $s$  a un nodo soluzione.

*Così è stato dimostrato che  $g(l) \leq g(l')$ , cioè che la soluzione  $l$  scoperta è **ottima**.*

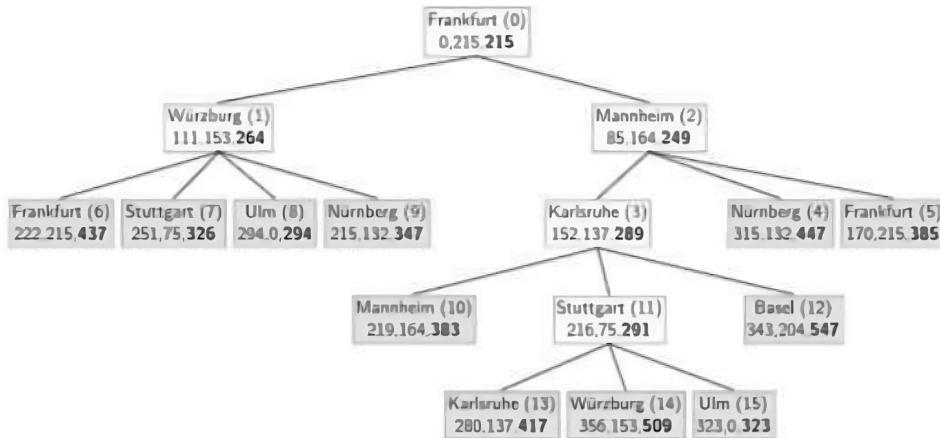
$$g(l) = g(l) + h(l) = f(l) \leq f(s) = g(s) + h(s) \leq g(l')$$

## 7.5 Esempio: applicazione A\*

*Cerchiamo il percorso più breve da Frankfurt a Ulm:*

Nelle caselle sotto il nome della città  $s$  sono indicati i valori di  $g(s), h(s), f(s)$ . I numeri tra parentesi dopo i nomi delle città mostrano l'ordine in cui i nodi sono stati generati dalla funzione Successor

$g(s), h(s), f(s)$



In questo caso non avviene l'ordinamento dei Successor. Vado ad ampliare prima Mannheim perché  $f(Mannheim) < f(Wurzburg)$ . I Successor(Mannheim) hanno tutti  $f < f(Wurzburg)$  quindi espando Wurzburg. E così via. Alla fine ci assicuriamo che la soluzione ottima viene data dalla (8).

## 7.6 Ricerca con avversari

- Se all'interno del problema abbiamo altri agenti (avversari), le nostre scelte dipenderanno anche da ciò che compie l'avversario.
- La scienza che studia tali interazioni è la **teoria dei giochi**. La teoria matematica dei giochi studia le interazioni tra più agenti che possono essere: **cooperativi o avversari**.
- I giochi più comuni analizzati dall'IA sono giochi a somma zero, ad informazione perfetta, a turni e a due giocatori.
- Gioco a somma zero o somma costante: alla fine di ogni partita la somma delle utilità per i giocatori è la stessa (noti anche come giochi "MAX MIN") un gioco nel quale ciò che un partecipante vince viene perso dall'altro. Per convenzione il **primo giocatore è MAX** e il **secondo è MIN**.

**Somma zero:** se il giocatore A vince 100 euro contro il giocatore B, quest'ultimo perderà esattamente 100 euro ( $\text{vincite} - \text{perdite} = 0$ ).

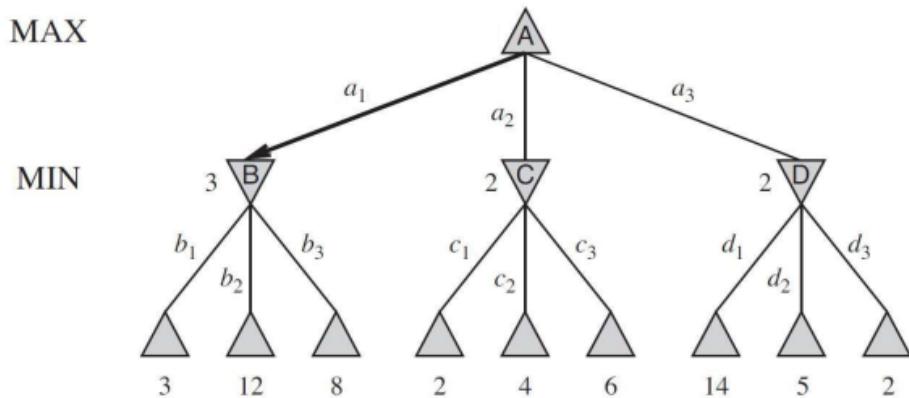
- Definizione di un gioco
  - **Stato iniziale:** configurazione iniziale del gioco
  - **Giocatori:** set dei giocatori e turni di gioco dei giocatori
  - **Azioni:** ammissibili per il giocatore attivo nello stato
  - **Risultato o modello di transizione:** come evolve il gioco a fronte di una data azione in un dato stato
  - **Test-terminazione:** definisce se uno stato è terminale, ovvero la partita ha termine
  - **Utilità:** definisce il valore di ogni giocatore in uno stato terminale
- **Le azioni di un giocatore devono essere contingenti:** mossa per stato iniziale (se iniziamo) e poi una mossa per ogni possibile comportamento dell'avversario.
- Parliamo allora di **strategia**: risposta per ogni mossa dell'avversario. **Strategia ottima:** strategia che porta un'utilità maggiore o uguale ad ogni altra possibile strategia, supponendo l'avversario "infallibile".

## 7.7 Gioco a singolo turno

**NB:** in questo gioco ho solo due strati che corrispondono ad una singola mossa per giocatore. Supponiamo inoltre che le giocate avvengano sequenzialmente.

**Funzionamento:**

- ci sono solo **2 mosse** una del giocatore **MAX** (il cui obiettivo è quello di **massimizzare** il punteggio) e una del giocatore **MIN** (il cui obiettivo è di **minimizzare** il punteggio).
- I nodi  $\Delta$  dono quelli in cui tocca Max, i nodi  $\nabla$  sono quelli in cui tocca a MIN
- Da un nodo partono diversi percorsi, ciascuno corrispondente a una mossa che potrebbe venire eseguita.
- Le foglie sono le configurazioni terminali del gioco, quando nessuna mossa è più possibile e a cui corrisponde un punteggio che stabilisce chi ha vinto e chi ha perso.



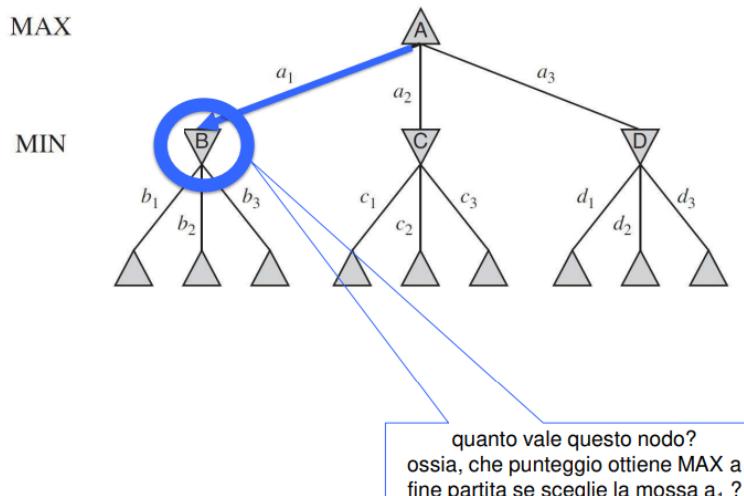
## 7.8 Algoritmo MINIMAX

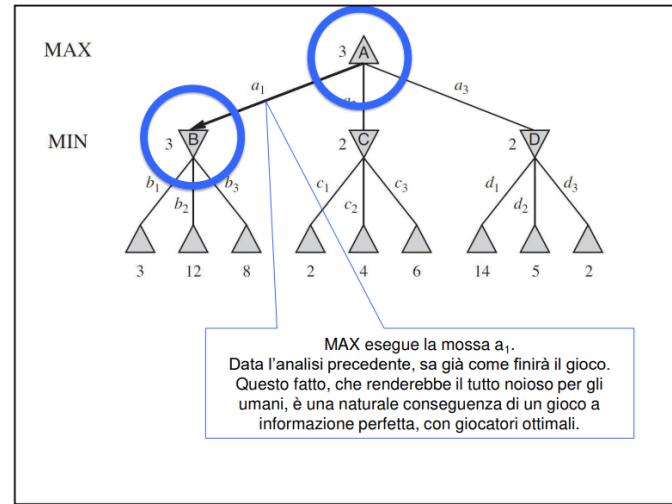
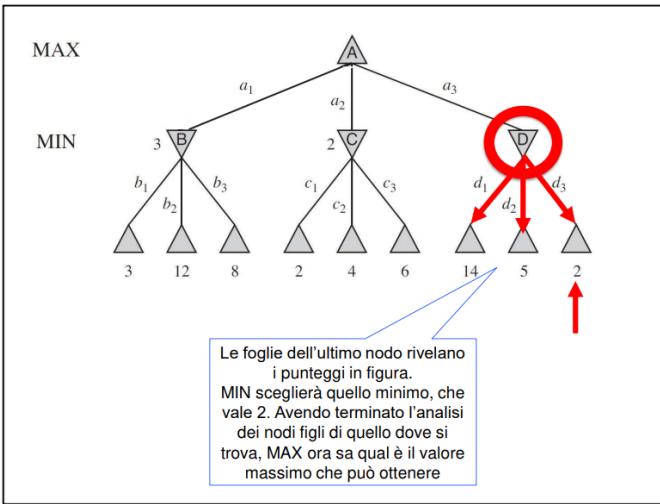
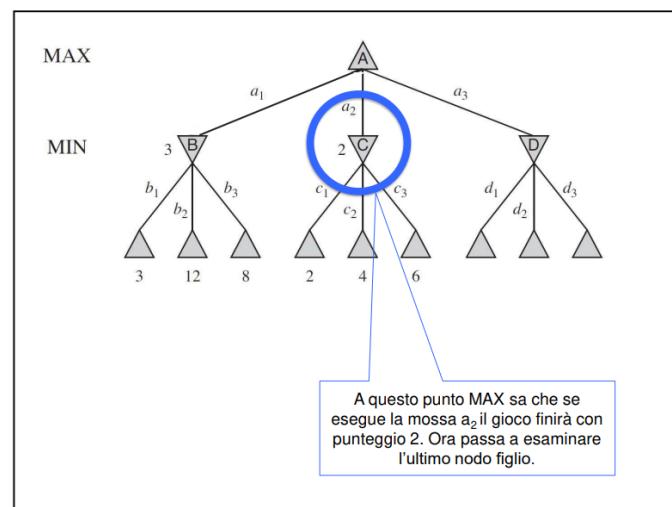
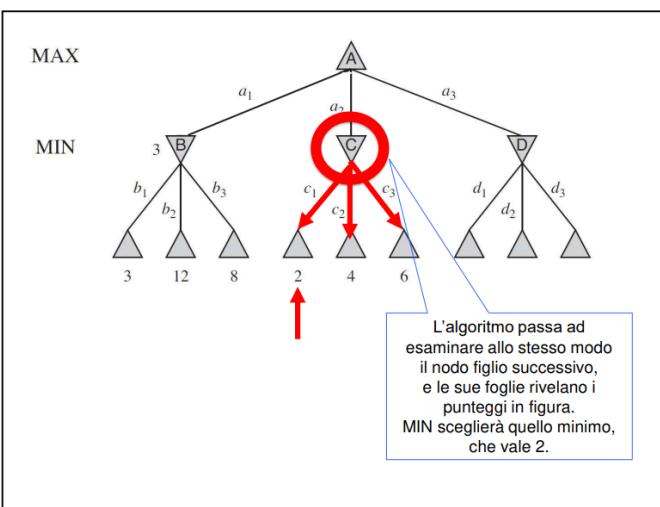
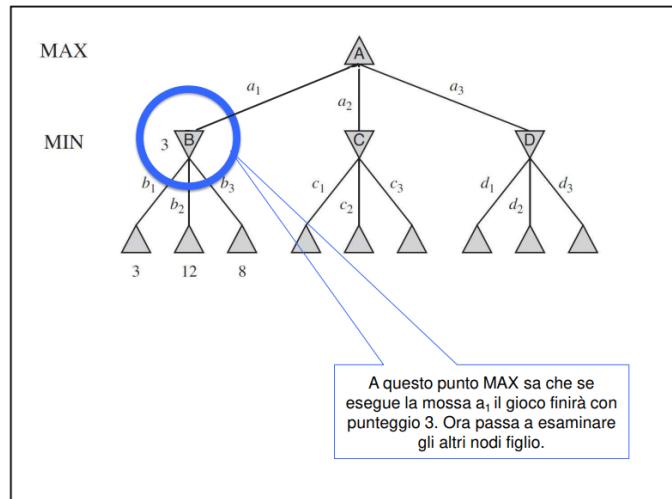
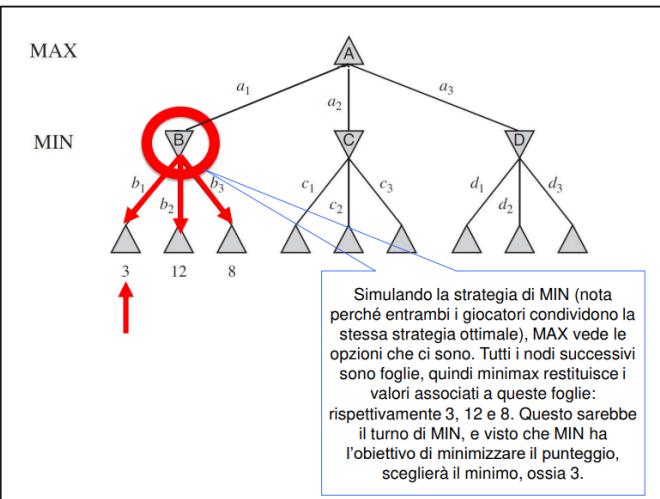
- **minimax(radice, TRUE):** chiamata iniziale, inizia MAX
- **minimax(node, MAX),** node nodo in cui si trova l'esecutore, che lancia **minmax()** per capire quale è la mossa migliore. MAX è un bool che è vero se il turno è di MAX, falso se è di MIN.
- **return util(node):** Se il nodo è una foglia, il gioco è finito e il valore del nodo è il risultato finale
- **v = max(v, minimax(child, FALSE)):** per ogni nodo figlio (futuri stati), calcola il suo valore simulando la strategia di MIN, e scegli il valore massimo
- **v = min(v, minimax(child, TRUE)):** questa è la parte eseguita da MIN, che cerca il nodo con il valore minimo, simulando la strategia di MAX

```

function minimax(node, MAX)
  if isLeaf(node) == true then
    return util(node)
  if MAX == true then
    v = -∞
    for each node.child do
      v = max(v, minimax(child, FALSE))
    return v
  else
    v = +∞
    for each node.child do
      v = min(v, minimax(child, TRUE))
    return v

```



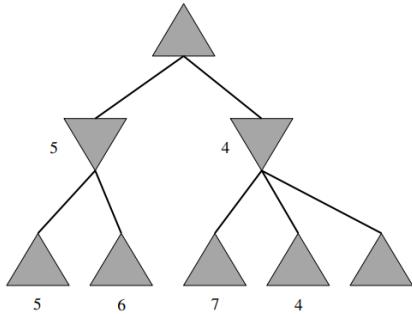


### Osservazioni:

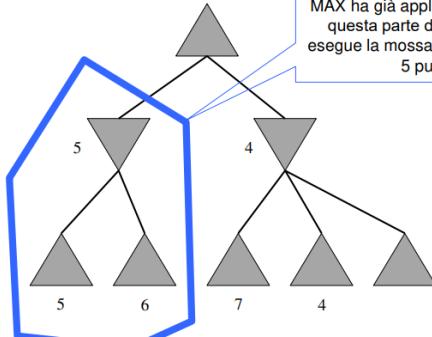
- Calcolo ricorsivo del valore di uno stato a partire dai nodi successori.
- La ricorsione si ferma alle foglie dove il valore è definito (e dato dal problema stesso).
- **Si ha completezza:** si riesce sempre a costruire una strategia da utilizzare.
- **Si ha ottimalità:** si ottiene sempre il risultato migliore compatibilmente con le regole del gioco e i valori delle foglie.
- Se  $b$  è il branching factor e  $d$  è la profondità massima dell'albero, la complessità temporale è  $O(b^d)$  e quella spaziale è  $O(b \cdot d)$

## 7.9 Potatura

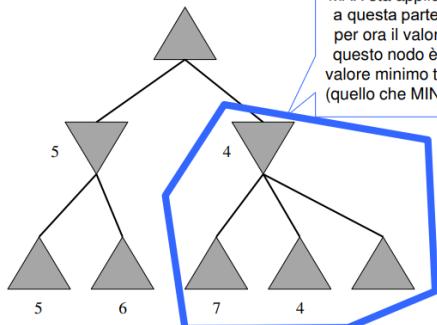
### Idea di base



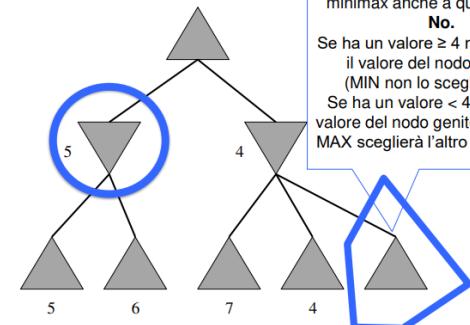
### Idea di base



### Idea di base



### Idea di base



### Osservazioni:

- Modifica di MINIMAX per non espandere inutilmente nodi dell'albero di ricerca che non porterebbero a una soluzione ottimale.
- L'algoritmo gestisce due valori  $\alpha$  e  $\beta$
- $\alpha$ : il miglior punteggio per MAX sinora trovato sul suo percorso
- $\beta$  : il miglior punteggio per MIN sinora trovato sul suo percorso
- I valori iniziali sono  $\alpha = -\infty$  e  $\beta = +\infty$  entrambi i giocatori iniziano col rispettivo peggior punteggio
- **Potatura per MIN:**  $\text{valore nodo corrente} \leq \alpha$ , MAX non farà mai arrivare il gioco lì perché ha altri percorsi migliori, quindi inutile proseguire con i controlli (esempio sopra)
- **Potatura per MAX:**  $\text{valore nodo corrente} \geq \beta$ , MIN non farà mai arrivare il gioco lì perché ha altri percorsi migliori, quindi inutile proseguire con i controlli

### Algoritmo:

- $\alpha$  viene aggiornato solo dal codice di MAX
- $\beta$  viene aggiornato solo dal codice di MIN
- I valori di  $\alpha$  e  $\beta$  vengono passati **dai nodi ai loro figli**, ma non viceversa
- I figli passano ai nodi genitori solo i valori dei nodi e non  $\alpha$  e  $\beta$
- La potatura, in termini di codice, vuol dire abbandonare l'analisi di tutti i nodi figli del nodo corrente, il che equivale a non analizzare un sottoalbero

Codice:

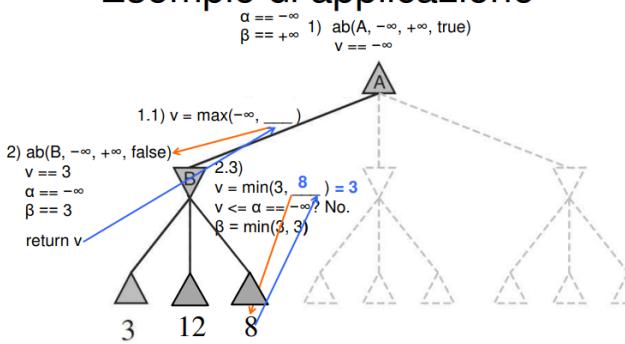
- Chiamata iniziale: alphabeta(radice, -inf, +inf, TRUE)
- **return util(node):** Se il nodo è una foglia, il gioco è finito e il valore del nodo è il risultato finale
- **if MAX == true then:** Questa è la parte di MAX. Dobbiamo calcolare il valore di un nodo non terminale. Si parte sempre dal valore peggiore possibile per MAX, a migliorare a seconda di quello che si trova nei nodi figli (ricorsivamente, simulando il comportamento di MIN). Se il valore trovato  $v \geq \beta$  interrompe il ciclo (potatura) e restituisce il valore del nodo. Altrimenti eventualmente aggiorna  $\alpha$  se il valore trovato lo migliora.

```

function alphabeta(node,  $\alpha$ ,  $\beta$ , MAX) is
  if isLeaf(node) == true then
    return util(node)
  if MAX == true then
    v = - $\infty$ 
    for each node.child do
      v = max(v, alphabeta(child,  $\alpha$ ,  $\beta$ , FALSE))
      if v >=  $\beta$  then
        break
       $\alpha$  = max( $\alpha$ , v)
    return v
  else
    v = + $\infty$ 
    for each node.child do
      v = min(v, alphabeta(child,  $\alpha$ ,  $\beta$ , TRUE))
      if v <=  $\alpha$  then
        break
       $\beta$  = min( $\beta$ , v)
    return v

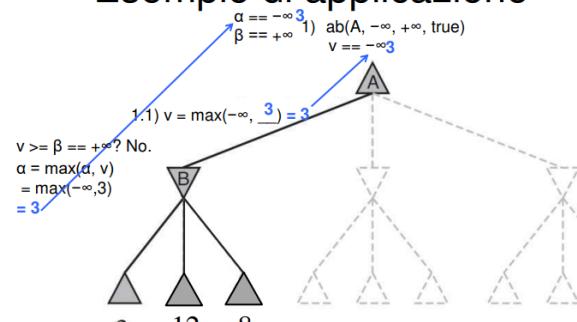
```

### Esempio di applicazione



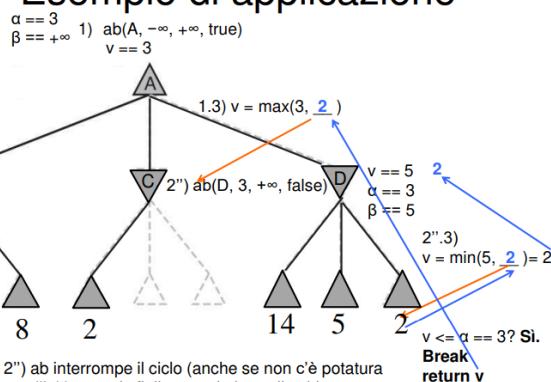
Fatta l'analisi dell'ultimo nodo figlio, la funzione 2) ab eseguita sul nodo B termina e restituisce  $v$ , che vale 3.

### Esempio di applicazione



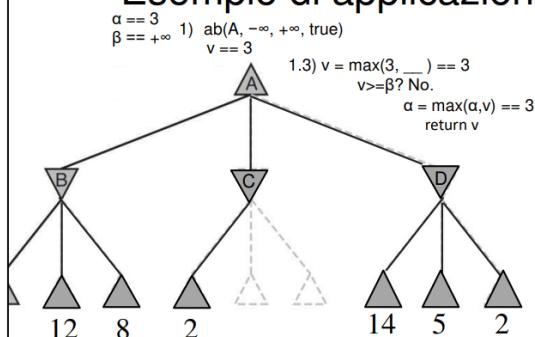
Ottenuto il risultato da 2) ab, la funzione 1) ab, eseguita al nodo A, può proseguire e aggiornare il valore di  $v$ . La condizione di potatura non è verificata, quindi passa a aggiornare  $\alpha$  dal valore iniziale  $-\infty$  al valore di  $v$ , ossia 3. Quindi la funzione 1) ab prosegue nel ciclo e esamina un altro nodo figlio (se c'è).

### Esempio di applicazione



La funzione 2') ab interrompe il ciclo (anche se non c'è potatura perché siamo sull'ultimo nodo figlio e restituisce alla chiamante 1) ab il valore di  $v = 2$ . La funzione 1) ab può riprendere la sua esecuzione.

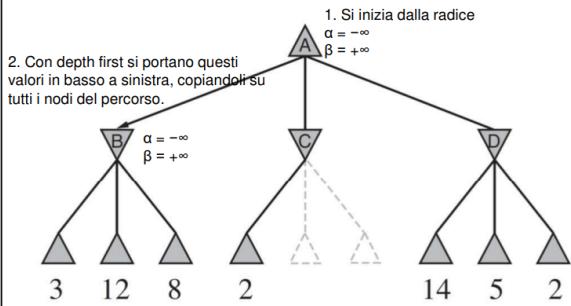
### Esempio di applicazione



La funzione 1) ab conclude restituendo il valore del nodo radice, che è 3, perché la strategia migliore per MAX è fare la mossa che lo porti al nodo B, una strategia che porterà alla conclusione del gioco con un punteggio di 3.

## Metodo che non segue il programma, ma più semplice

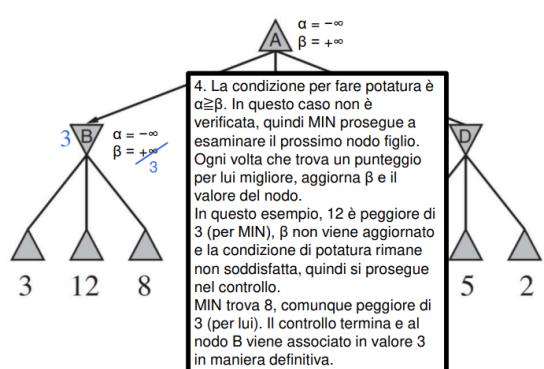
Metodo che non segue il programma, ma più semplice



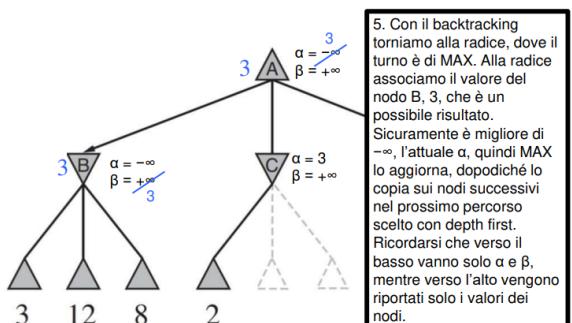
Metodo che non segue il programma, ma più semplice

3. Arrivati davanti a un nodo foglia, ci chiediamo a chi tocca.  
 Solo MAX, che vuole massimizzare il punteggio, può modificare  $\alpha$ .  
 $\alpha$  viene modificato se MAX trova una mossa che gli garantisce un punteggio più alto di  $\alpha$  (migliore per MAX).  
 Solo MIN, che vuole minimizzare il punteggio, può modificare  $\beta$ .  
 $\beta$  viene modificato se MIN trova una mossa che gli garantisce un punteggio più basso di  $\beta$  (migliore per MIN).  
 In questo esempio tocca a MIN, quindi assumiamo il suo punto di vista.  
 Esaminando i nodi figli da sinistra a destra, MIN trova innanzitutto 3.  
 3 è meglio dell'attuale  $\beta$ , quindi MIN aggiorna  $\beta$ . Il nodo B ha valore 3 (al momento, almeno).

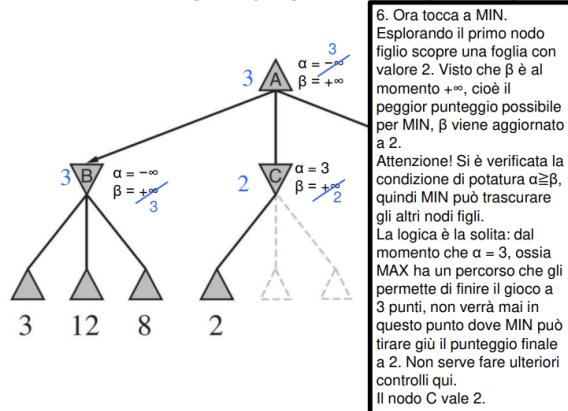
Metodo che non segue il programma, ma più semplice



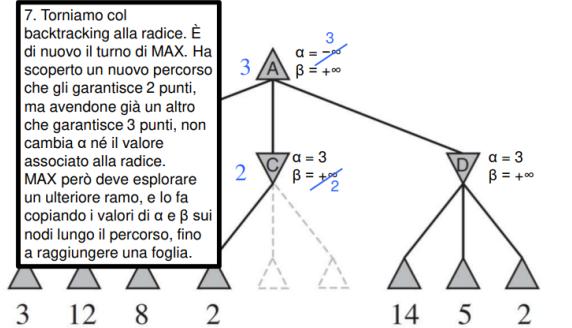
Metodo che non segue il programma, ma più semplice



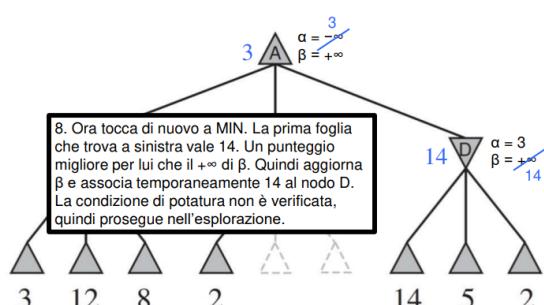
Metodo che non segue il programma, ma più semplice



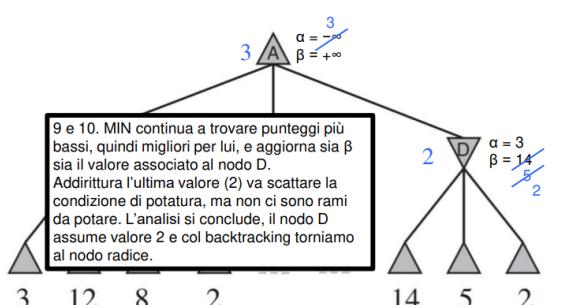
Metodo che non segue il programma, ma più semplice



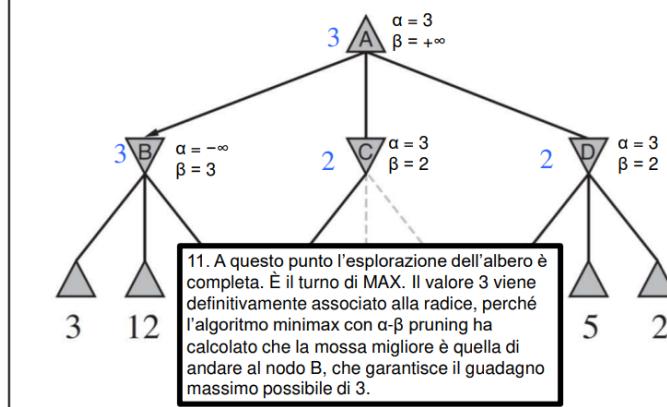
Metodo che non segue il programma, ma più semplice



Metodo che non segue il programma, ma più semplice



### Metodo che non segue il programma, ma più semplice



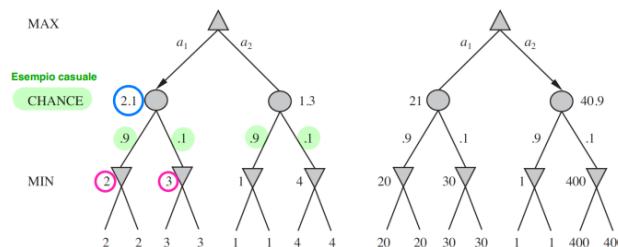
#### Caratteristiche e problemi della potatura:

- Quanto si riesce a potare dipende dall'ordine con cui visitiamo i nodi
- Con una buona euristica di ordinamento dei nodi la complessità temporale può scendere a un minimo di  $O(b^{\frac{d}{2}})$
- Nei casi reali arrivare a calcolare i valori degli stati finali (nodi foglia) non è pratico dal punto di vista computazionale e serve introdurre delle funzioni di valutazione del valore degli stati intermedi, per cercare di fare dei tagli di interi sottoalberi non promettenti

## 7.10 Giochi non deterministici

- I giochi contengono una componente non deterministica/stocastica: incontrollabile da parte degli agenti
- Nel modello ad albero introduciamo nodi che rappresentano questo tipo di eventi (**chance events**)
- Non si calcola più il valore esatto di una strategia intera, ma il valore atteso di una mossa (MINIMAX diventa **expectiMINIMAX**)

### Presupposizioni sulla funzione di valutazione



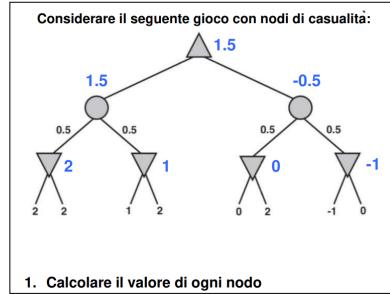
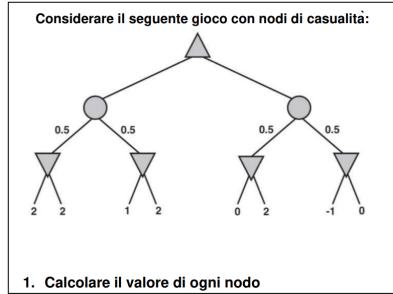
La funzione deve essere una trasformazione lineare della probabilità di vincere data una posizione

$$0,9 \cdot 2 + 0,1 \cdot 3 \Rightarrow \text{medie pesate}$$

$$1,8 + 0,3 = 2,1$$

*Considerare il seguente gioco con nodi di casualità:  
rispondere alle seguenti domande*

1. Calcolare il valore di ogni nodo
2. Conosciuti i valori dei primi 6 nodi foglia (partendo da sinistra), posso potare gli ultimi due nodi?
3. E se conoscessi che il range di valori di utilità nella foglie è nell'intervallo  $[-2, 2]$ ?
4. Avrei potuto anche non valutare altre foglie?



*Risposte:*

2. Se fossi sicuro che  $0.5x < 1.5$ , sì. Ma non lo sono. Con un  $x$  alto a piacere il ramo dx > ramo sx.
3. In tal caso sì: al massimo si ha  $x = 2$ , e MAX va di sicuro a sinistra.
4. Scoprendo uno 0 sulla prima foglia nel sottoalbero di destra, ogni altra foglia di quel sottoalbero può essere trascurata

## 8 Constraint Satisfaction problem

- Fino ad ora ogni stato è stato modellizzato come una **black box** a cui si poteva applicare una funzione di valutazione per verificare se fosse l'obiettivo, oppure per calcolare una stima del costo per arrivare da quello stato all'obiettivo finale
- Proviamo ora a considerare una **rappresentazione fattorizzata** per ogni stato, in cui ogni stato è rappresentato da un insieme di variabili con valore.
- Un problema è risolto quando tutti i valori dati alle variabili soddisfano un set di vincoli.
- elimino delle porzioni di spazio di ricerca che sicuramente non soddisfano i vincoli.

*Stato → black box → rappresentazione fattorizzata (insieme di variabili)*

### 8.1 Definizione del problema

**Definizione:** Un Constraint Satisfaction Problem è definito completamente da:

- Un insieme di variabili
- Un insieme di domini (uno per ogni variabile)
- Un insieme di vincoli che determinano valori di variabili ammissibili

**Ogni vincolo** è costituito da: l'insieme di variabili considerate nel vincolo e la relazione che devono soddisfare.

*Esempio: è un vincolo che specifica che le due variabili devono avere valori differenti*

$$\langle \{V_1; V_2\}, V_1 \neq V_2 \rangle$$

**Definizione:** Stato: assegnamento di valori ad alcune variabili. Soluzione: assegnamento completo e consistente con i vincoli.

Il CSP, o **problema di soddisfacimento dei vincoli**, può essere definito come una search tra stati, in cui:

- stato: assegnamento di valori ad alcune variabili (non necessariamente tutte)
- soluzione: assegnamento completo delle variabili consistente con i vincoli dati

*Esempio: ogni regione confinante deve avere un colore diverso, per questo motivo non compare la Tasmania, perché non confina con nessun stato.*

#### Es: colorare la mappa dell'Australia

Variabili:  
 $X = \{WA, NT, Q, SA, NS, V, T\}$

Dominio:  
 $D_i = \{r, v, b\}$

Vincoli:  
 $SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NS, SA \neq V, WA \neq NT, WA \neq Q, WA \neq NS, WA \neq V$

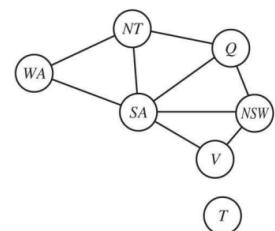


#### Grafo dei vincoli

Trasformiamo il problema nel seguente modo:

- Ogni variabile è un nodo
- Ogni arco esiste se i nodi sono presenti in almeno un singolo vincolo

ES: Australia



*Considerare questo come un problema di ricerca significa valutare  $3^7 = 2187$  (7 territori, 3 colori) differenti assegnamenti completi delle variabili*

## 8.2 Varianti del CSP

- **Dominio discreto ma infinito:** non posso più enumerare i valori ammissibili, devo definire un linguaggio di specifica di vincoli
- **Dominio discreto, ma infinito con vincoli lineari:** programmazione lineare intera
- **Dominio continuo, vincoli lineari:** programmazione lineare

Tipi di vincoli:

<i>Unario</i>	<i>es: </i> $\langle\{SA\}, SA \neq v\rangle$
<i>Binario</i>	<i>es: </i> $\langle\{SA; NS\}, SA \neq NS\rangle$
<i>Globale (tutte le var)</i>	<i>es: </i> <i>var tutte diverse AllDiff</i>

## 8.3 Inferenza nei CSP

- L'idea è quella di limitare il numero di valori che sono potenzialmente assegnabili alle variabili
- **Constraint propagation** (propagazione di vincoli): se restringo i valori assegnabili a una variabile, forse si restringono anche quelli assegnabili ad altre variabili
- Tale propagazione avviene per mezzo di ragionamenti assimilabili a **regole di inferenza** → *consistenza*
- Esistono diversi tipi di consistenza:
  - **Consistenza di nodo** → vincoli unari (restrizione sul dominio)
  - **Consistenza di arco** → vincoli binari (archi, restrizioni 2 domini su relazione arco)
  - **Consistenza di cammino (path)** → vincoli su 3 variabili (cammino)
  - **k-consistenza**

### Consistenza di nodo

- Consideriamo una variabile e il nodo del grafo che le corrisponde
- **La variabile ha consistenza di nodo** se il suo dominio viene ristretto in modo tale da soddisfare tutti i suoi vincoli unari
- **Ese:** se abbiamo  $\langle\{SA\}, SA \neq v\rangle$  riducendo il dominio di  $SA$  da  $\{r, v, b\}$  a  $\{r, b\}$  otteniamo consistenza di nodo

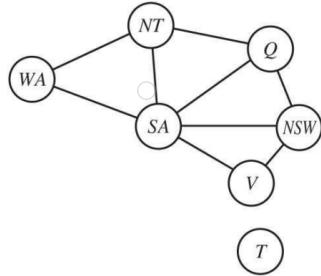
### Consistenza di arco

- **Gli archi** che partono da un nodo rappresentano vincoli per la variabile corrispondente al nodo
- **La variabile ha consistenza di arco** se il suo dominio è fatto di valori che soddisfano tutti i vincoli binari della variabile → **arc-consistent**
- **Ese:** se il CSP consiste in due variabili  $X$  e  $Y$ , con dominio comune  $\{0; 1; \dots; 9\}$  e con vincolo  $Y = X^2$  ossia  $\Rightarrow \langle(X, Y), \{(0, 0); (1, 1); (2, 4); (3, 9)\}\rangle$  perché  $X$  abbia consistenza di arco, il suo dominio  $D_X$  deve diventare  $\{0; 1; 2; 3\}$ , se anche  $Y$  acquista consistenza di arco, ovvero  $D_Y = \{0; 1; 4; 9\}$  allora l'intero CSP è **arc-consistent**

### Consistenza di cammino (path)

- Estensione vincoli 3 variabili
- Un insieme di 2 variabili  $\{X_i; X_j\}$  ha **consistenza di cammino** con una terza variabile  $X_m$  se per ogni assegnamento  $\{X_i = a; X_j = b\}$  che rispetta i vincoli su  $\{X_i; X_j\}$  esiste un assegnamento a  $X_m$  che soddisfa i vincoli sia su  $\{X_i; X_j\}$  sia su  $\{X_j; X_m\}$
- **NB:** L'idea è di creare un cammino da  $X_i$  a  $X_j$  via  $X_m$

Cerchiamo di dare a  $\{WA; SA\}$  consistenza di cammino via  $NT$ .



Considerato che c'è il vincolo unario  $\langle(SA), SA \neq v\rangle$  e il vincolo binario  $SA \neq WA$ , e considerati i vincoli  $WA \neq NT$  e  $SA \neq NT$  gli unici assegnamenti consistenti sono:

$$\{WA = r; SA = b\} \text{ e } \{WA = b; SA = r\}$$

## K-Consistenza

- Un CSP si dice **k-consistente** se per ogni insieme di  $k - 1$  variabili e per ogni assegnamento consistente per tali  $k - 1$  variabili esiste sempre un valore consistente per la  $k - esima$  variabile
- Un CSP si dice **fortemente k-consistente** se è  $(k - 1) - consistente$ ,  $(k - 2) - consistente$ , ..., fino a  $2 - consistente$  e  $1 - consistente$

## 8.4 Algoritmo AC3

- AC3 serve per verificare la consistenza di tutti gli archi di un CSP
- AC3 = "Algoritmo consistente 3° terzo algoritmo presentato nel paper di A. Mackworth"
- function AC3 (csp) returns bool: in output, AC3 restituisce TRUE se csp è consistente, FALSE altrimenti. In input riceve un csp binario (ossia con soli vincoli binari) con componenti  $(X, D, C)$ : variabili, i loro domini, vincoli binari

```

function AC3 (csp) returns bool
  while q non vuota do
     $(X_i, X_j) \leftarrow REMOVE\_1ST(q)$ 
    if REVISE(csp,  $X_i, X_j$ ) then
      if SIZEOF( $D_i$ ) == 0 then
        return FALSE
      for each  $X_k$  in  $X_i.\text{ADJ} \setminus \{X_j\}$  do
        add( $X_k, X_i$ ) to q
    return TRUE
  
```

$$\text{csp} \equiv (X, D, C) = (\text{variabili, dom var, vincoli binari})$$

- q var locale: una coda di archi di csp con valore iniziale l'insieme di tutti gli archi in csp
- $(X_i, X_j) \leftarrow REMOVE_1ST(q)$ : si toglie da q il primo arco disponibile e lo si assegna alla variabile locale strutturata
- REVISE: è una funzione che restringe il dominio di  $X_i$  per far rispettare i suoi vincoli con  $X_j$ . Resutuisce TRUE quando c'è stato un restrinimento. FALSE vuol dire che il dominio è compatibile con tutti i vincoli e siamo a posto.
- SIZEOF: se REVISE ha ridotto il dominio di  $X_i$  all'insieme vuoto, vuol dire che i vincoli  $(X_i, X_j)$  non possono essere soddisfatti
- for each: ciclo for su tutti i nodi adiacenti a  $X_i$  escluso  $X_j$  e visto che c'è stato un restrinimento di  $D_i$  per rispettare i vincoli con  $X_j$  dobbiamo ricontrollare i vincoli con tutti gli altri vicini di  $X_i$
- RETURN TRUE: q vuota, ossia tutti i vincoli controllati sono OK

```

function REVISE (csp,  $X_i, X_j$ ) returns bool
  bool revised  $\leftarrow$  FALSE
  for each x in  $D_i$  do
    if  $\nexists y \in D_j : (x, y)$  soddisfa il vincolo  $(X_i, X_j)$ 
      then  $D_i \leftarrow D_i \setminus \{x\}$ 
      revised  $\leftarrow$  TRUE
  return revised
  
```

- function REVISE() return bool: In input arrivano il csp e due variabili. L'output è TRUE se il dominio di  $X_i$  ha dovuto restringersi per rispettare i vincoli con  $X_j$ .
- for each: Il ciclo di controlli si svolge su ciascun valore nel dominio  $D_i$ . Dato il valore  $x$  in  $D_i$ , se non si trova un valore  $y$  in  $D_j$  per cui  $(x, y)$  soddisfa il vincolo  $x$  va eliminato.

## 8.5 Esempio: Sudoku

Il sudoku è un CSP con 81 variabili ( $A_1, A_2, \dots, I_8, I_9$ ). I domini delle celle vuote sono tutti  $\{1, \dots, 9\}$ . I domini delle celle precompilate contengono un solo valore. Ci sono 27 vincoli AllDiff:

- 9 righe: AllDiff( $A_1, A_2, \dots, A_9$ ), AllDiff( $B_1, B_2, \dots, B_9$ ), ...
- 9 colonne
- 9 unità (sarebbe i quadrati)
- I vincoli AllDiff si possono ridurre a vincoli binari:  $A_1 \neq A_2, A_2 \neq A_3, \dots$

	1	2	3	4	5	6	7	8	9
A			3		2	6			
B	9			3	5				1
C			1	8	6	4			
D		8	1		2	9			
E	7								8
F		6	7		8	2			
G			2	6	9	5			
H	8			2	3				9
I		5		1		3			

	1	2	3	4	5	6	7	8	9
A			3		2	1	6		
B	9			3	5				1
C			1	8	6	4			
D		8	1		2	9			
E	7				4				8
F		6	7		8	2			
G		2	6		9	5			
H	8			2	3				9
I		5		1	7	3			

### Uso di AC3 per calcolare E6

Il dominio iniziale è come segue:  $D_{E6} = \{1; 2; 3; 4; 5; 6; 7; 8; 9\}$ . Focalizziamoci sulle chiamate a REVISE tra E6 e:

- i suoi adiacenti di **colonna**, con cui vale AllDiff( $A_6, \dots, I_6$ ), restringiamo  $D_{E6} = \{1; 4; 7\}$
- i suoi adiacenti di **riga**, con cui vale AllDiff( $E_1, \dots, E_9$ ), restringiamo ulteriormente  $D_{E6} = \{1; 4\}$
- con il vincolo di unità AllDiff( $D_4, D_5, D_6, E_4, E_5, E_6, F_4, F_5, F_6$ ) il dominio si restringe a un solo valore  $D_{E6} = \{4\} \Rightarrow$  Risolto

## 8.6 Search in CSP

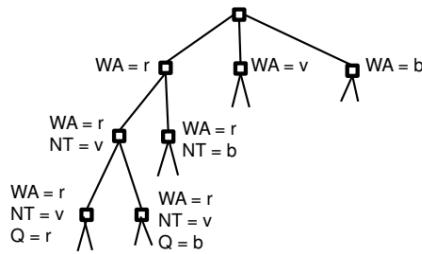
- Siamo arrivati a  $D_{E6} = \{4\}$  per mezzo di inferenze basate sui vincoli.
- Alcuni CSP non possono essere risolti con la sola inferenza: può servire andare a tentativi e tornare indietro quando si trovano inconsistenze.
- Questi tentativi ci riportano alla **search in una struttura ad albero**, dove il branching è dato dalle diverse mosse possibili
- Per un CSP con: **n variabili**, ciascuna con **domini di dimensione d**, il **branching factor** iniziale è di  $n \times d$  (la scelta è tra qualunque valore da assegnare a qualunque variabile). Nel livello successivo avremo  $(n - 1) \times d$  poi  $(n - 2) \times d$  e così via.

Variabili	$n$
Domini var:	$d$
Branching factor	$n \times d$

- Questo calcolo mostra che ci sono  $n! \cdot d^n$  foglie, ma in realtà ci sono solo  $d^n$  **assegnamenti possibili** di  $d$  valori a  $n$  variabili con possibili ripetizioni.  $n! \cdot d^n$  salta fuori se ci concentriamo sui vari path dell'albero, ma alcuni di questi path che portano allo stesso assegnamento finale e tale differenza deriva solo dall'ordine con cui scegliamo le variabili a cui assegnare valori
- L'ordine con cui assegno i valori
  - non influenza il risultato finale  $\rightarrow d^n$  possibili risultati diversi
  - bensì influenza il modo in cui vi arrivo  $\rightarrow n! \cdot d^n$  possibili path
- L'ordine non conta nei CSP, quindi possiamo restringere l'albero considerando solo una variabile per livello

## 8.7 Esempio: mappa Australia

- In questo albero decidiamo di occuparci prima della variabile WA, poi di NT, poi di Q, etc
- Procedendo depth-first, si fa l'assegnamento a una variabile per volta e se si scopre un'inconsistenza, si torna indietro con backtracking
- Nel procedere con la search, però, bisogna eseguire numerose scelte:
  - Scelta dell'ordine di elaborazione delle variabili
  - Scelta dell'ordine di assegnamento dei valori a una variabile
  - A ogni nodo dell'albero, scegliere se fare inferenza o meno
  - Sempre e comunque, evitare assegnamenti che violano i vincoli



## 8.8 Strategie

### Scelta della variabile 1

- Si sceglie la variabile con meno valori a disposizione, così se fallisco, fallisco presto. Euristica basata sul criterio **fail fast**
- Euristica: **MRV (Minimum Remaining Values)**, sperimentalmente si è verificato che in media MRV è più efficiente di una scelta random della variabile → dominio più piccolo
- Questa strategia, però, è a volte problematica all'inizio della soluzione di un CSP

### Scelta della variabile 2

- Si sceglie la variabile con maggiori vincoli, il che può ridurre il branching factor di scelte future
- **Euristica di grado (Degree heuristics)**
- In generale: MRV è utile e potente, Degree utile per risolvere situazioni di pareggio

### Scelta della variabile 3

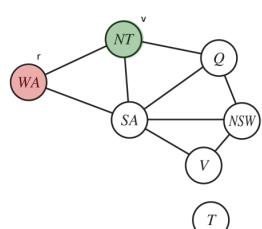
- ORD: scelgo la variabile per ordinamento Alfabetico
- RND: scelgo a caso la var

### Scelta del valore

- Una volta scelta la variabile, quale dei valori del suo dominio le assegniamo?
- Scelta 1: per massimizzare la probabilità di trovare una soluzione, sceglio il **valore meno vincolante**, quello che lascia più libere le variabili adiacenti nel grafo ⇒ Euristica: **LCV (Least Constraining Value)**
- Scelta 2: **ORD** scelgo il valore in base al ordinamento del Dominio

*Esempio:*

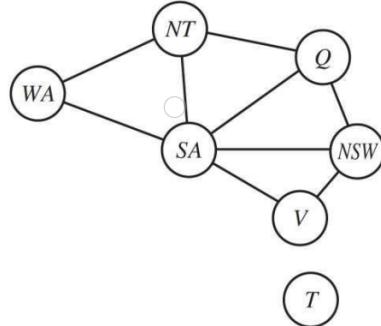
- Supponiamo di avere già  $WA = r$  e  $NT = v$  e che la prossima var è  $Q$
- Considerati i vincoli su  $SA$ , non sceglio  $Q = b$  bensì  $Q = r$
- $Q = b$  eliminerebbe l'ultimo valore possibile per  $SA$



## 8.9 Forward checking (FC)

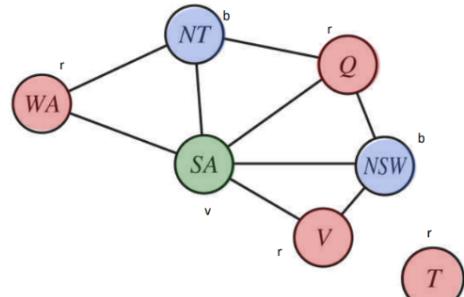
- A ogni passo della ricerca possiamo valutare se ci sono delle consistenze d'arco che escludono alcuni valori delle variabili, combinando ricerca e inferenza
- **Funzionamento** = ogni volta che a una variabile  $X$  si assegna un valore  $x$ , si stabilisce la consistenza d'arco per  $X$ : per ogni variabile  $Y$  senza valore e connessa a  $X$  da un vincolo, bisogna cancellare ogni valore del suo dominio  $D_Y$  che sia inconsistente con  $x$
- **Da solo FC combina poco**, supponendo che le mosse siano  $WA = r$ ,  $Q = v$ ,  $V = b$  controlliamo e riduciamo i domini delle variabili connesse a quella appena elaborata. **Finiamo in un vicolo cieco, niente valori per SA**

Domini	WA	NT	Q	NSW	V	SA	T
inizio	{r; v; b}	{r; v; b}	{r; v; b}	{r; v; b}	{r; v; b}	{r; v; b}	{r; v; b}
dopo $WA = r$	<b>== r</b>	{v; b}	{r; v; b}	{r; v; b}	{r; v; b}	{v; b}	{r; v; b}
dopo $Q = v$	<b>== r</b>	{b}	<b>== v</b>	{r; b}	{r; v; b}	{b}	{r; v; b}
dopo $V = b$	<b>== r</b>	{b}	<b>== v</b>	{r}	<b>== b</b>	<b>Ø</b>	{r; v; b}



- Applichiamo le strategie viste precedentemente:
  - **Scelta variabili:** MRV > Degree > ORD > RND
  - **Scelta valore:** LCV > ORD
  - Immediatamente dopo ogni assegnamento, aggiorniamo i domini delle variabili con FC
  - **Il valore all'interno di un dominio viene scelto seguendo l'ordinamento** ( $r, v, b$ )

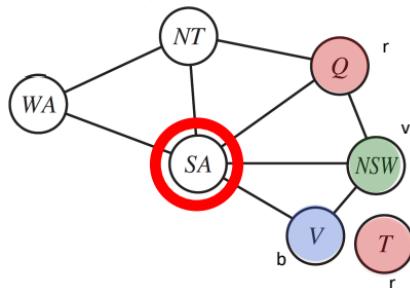
Domini	WA	NT	Q	NSW	V	SA	T	Commento
inizio	<b>[r; v; b]</b>	{r; v; b}	{r; v; b}	{r; v; b}	{r; v; b}	{r; v; b}	{r; v; b}	all'inizio tutte le variabili hanno domini completi
dopo $WA = r$	<b>== r</b>	{v; b}	{r; v; b}	{r; v; b}	{r; v; b}	<b>[v; b]</b>	{r; v; b}	FC restringe i domini di NT e SA. MRV non sa decidere, Degree sceglie SA. Il valore scelto è v per ordinamento.
dopo $SA = v$	<b>== r</b>	<b>[b]</b>	{r; b}	{r; b}	{r; b}	<b>== v</b>	{r; v; b}	FC restringe i domini di NT, Q, NSW e V. MRV sceglie NT.
dopo $NT = b$	<b>== r</b>	<b>== b</b>	<b>[r]</b>	{r; b}	{r; b}	<b>== v</b>	{r; v; b}	FC restringe il dominio di Q e MRV lo sceglie.
dopo $Q = r$	<b>== r</b>	<b>== b</b>	<b>== r</b>	<b>[b]</b>	{r; b}	<b>== v</b>	{r; v; b}	FC restringe il dominio di NSW e MRV lo sceglie.
dopo $NSW = b$	<b>== r</b>	<b>== b</b>	<b>== r</b>	<b>== b</b>	<b>[r]</b>	<b>== v</b>	{r; v; b}	FC restringe il dominio di V e MRV lo sceglie.



$T$  ha il dominio completo, scegliamo seguendo l'ordinamento quindi  $r$

- Supponendo che le mosse siano  $Q = r$ ,  $NSW = v$ ,  $V = b$ ,  $T = r$  e che ora tocchi a  $SA$ , **non ci sono valori ammissibili!** Facendo backtracking torniamo alla mossa precedente e assegniamo un nuovo valore a  $T$ , siamo comunque bloccati.

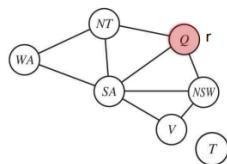
Tornare indietro con il backtracking in questo contesto non è sufficiente. Bisogna tornare indietro fino alla variabile che ha causato il problema.  $\Rightarrow$  **backjumping**. Salta direttamente alla mossa che ha causato il problema



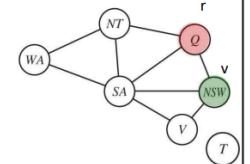
## 8.10 Backjumping e conflict set

- Per poter fare backjumping, dobbiamo individuare il punto di destinazione, nel caso precedente  $SA$
- Teniamo traccia dei possibili conflitti che possono generarsi dall'assegnamento  $\Rightarrow$  **conflict set**
- Ogni volta che eseguiamo un assegnamento  $X = x$  e conseguentemente con  $FC$  restringiamo il dominio di una variabile  $Y \rightarrow X = x$  viene aggiunto al conflict set di  $Y$ . Se il restringimento genera  $D_Y = \{\}$  tutti gli assegnamenti nel conflict set di  $Y$  vanno aggiunti al conflict set di  $X$

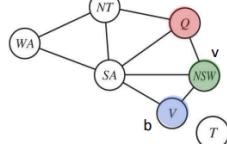
CS	WA	NT	Q	NSW	V	SA	T	Commento
Domini	{r; v; b}	/v; b)	{r; v; b}	/v; b)	{r; v; b}	/v; b)	{r; v; b}	
dopo $Q = r$		FC restringe i domini di NT, NSW e SA. L'assegnamento va inserito nei loro Conflict Set.						



CS	WA	NT	Q	NSW	V	SA	T	Commento
Domini	{r; v; b}	/v; b)	{r; v; b}	/v; b)	{r; v; b}	/v; b)	{r; v; b}	
dopo $Q = r$		FC restringe i domini di NT, NSW e SA. L'assegnamento va inserito nei loro Conflict Set.						

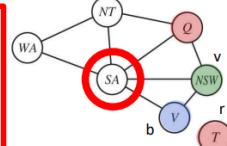


CS	WA	NT	Q	NSW	V	SA	T	Commento
Domini	{r; v; b}	/v; b)	{r; v; b}	/v; b)	{r; v; b}	/v; b)	{r; v; b}	
dopo $Q = r$		$Q = r$		$Q = r$		$Q = r$		FC restringe i domini di NT, NSW e SA. L'assegnamento va inserito nei loro conflict set.
dopo $NSW = v$			$NSW = v$		$NSW = v$	$Q = r$	$NSW = v$	FC restringe i domini di Q, V e SA. L'assegnamento va inserito nei loro conflict set.



CS	WA	NT	Q	NSW	V	SA	T	Commento
Domini	{r; v; b}	/v; b)	{r; v; b}	/v; b)	{r; v; b}	/v; b)	{r; v; b}	
dopo $Q = r$		$Q = r$		$Q = r$		$Q = r$		FC restringe i domini di NT, NSW e SA. L'assegnamento va inserito nei loro conflict set.
dopo $NSW = v$			$NSW = v$		$NSW = v$	$Q = r$	$NSW = v$	FC restringe i domini di Q, V e SA. L'assegnamento va inserito nei loro conflict set.
dopo $V = b$				$Q = r$	$V = b$	$Q = r$	$NSW = v$	FC restringe i domini di NSW e SA. L'assegnamento va inserito nei loro conflict set. Il dominio di SA è ora vuoto: il suo CS va unito a quello di V.
dopo $T = r$								Non succede niente perché non ci sono restrimenti
SA ??								

Ora che dobbiamo assegnare un valore a  $SA$  siamo bloccati perché il suo dominio è vuoto, ovvero ogni assegnamento andrebbe in conflitto con i vincoli. Si esegue il backjump: **si torna indietro al più recente assegnamento del conflict set di  $SA$ .**



CS	WA	NT	Q	NSW	V	SA	T	Commento
Domini	{r; v; b}	/v; b)	{r; v; b}	/v; b)	{r; v; b}	/v; b)	{r; v; b}	
dopo $Q = r$		$Q = r$		$Q = r$		$Q = r$		FC restringe i domini di NT, NSW e SA. L'assegnamento va inserito nei loro conflict set.
dopo $NSW = v$			$NSW = v$		$NSW = v$	$Q = r$	$NSW = v$	FC restringe i domini di Q, V e SA. L'assegnamento va inserito nei loro conflict set.
dopo $V = b$				$Q = r$	$V = b$	$Q = r$	$NSW = v$	FC restringe i domini di NSW e SA. L'assegnamento va inserito nei loro conflict set. Il dominio di SA è ora vuoto: il suo CS va unito a quello di V.
dopo $T = r$								Non succede niente perché non ci sono restrimenti
SA ??								

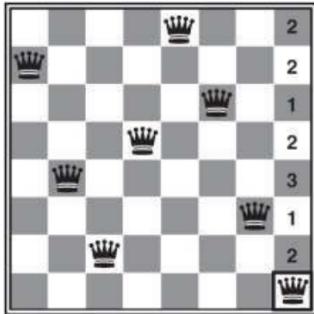
Grazie alla traccia tenuta con i conflict set, siamo in grado di individuare l'assegnamento che ha causato il problema che stiamo cercando di eliminare. Facciamo backjumping fino a  $V = b$  e proseguiamo la search con un assegnamento diverso ( $V = r$ )

## 8.11 Un altro approccio: ricerca locale

- **Idea:** parto da una configurazione non valida e mi muovo a poco a poco verso una valida
- Ricerca locale con euristica **min-conflicts**: vario il valore di una delle variabili per minimizzare il numero di violazioni di vincoli

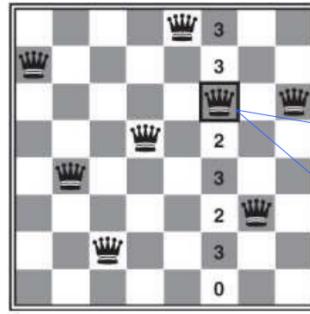
*Es: Il problema delle otto regine*  
*È il problema di piazzare 8 regine su una scacchiera 8x8*  
*in modo tale che nessuna possa mangiarne un'altra*

### Configurazione 1



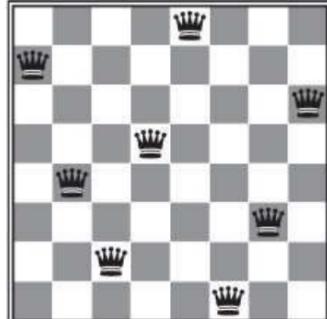
Questa regina è sotto attacco lungo la diagonale. La dobbiamo spostare. Gli spostamenti sono organizzati lungo la colonna. La regina in qualunque posizione rimane comunque sotto attacco. Scegliamo quella col minore numero di attacchi a partire dall'alto.

### Configurazione 2



Controlliamo le altre regine, colonna per colonna, da destra a sinistra. La seconda regina è a posto: non ci sono attacchi. La terza, invece, è sotto attacco proprio da quella che abbiamo spostato al turno precedente. Controlliamo le altre posizioni della terza colonna e ne scopriamo una senza attacchi. Spostiamo la terza regina lì.

### Configurazione 3

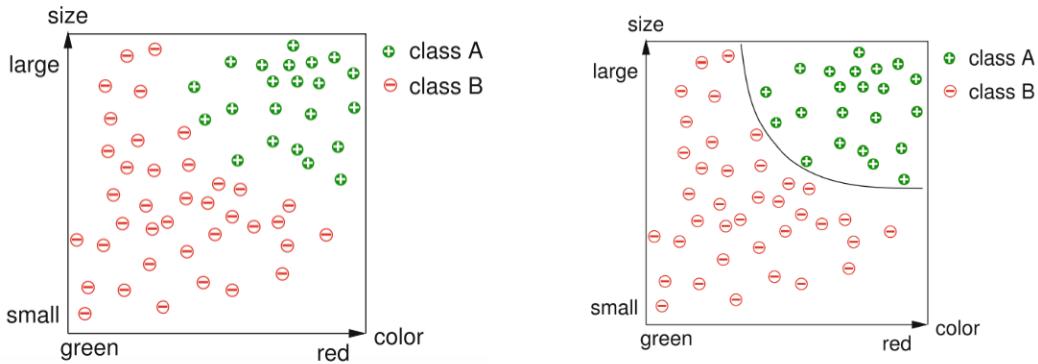


Con questo spostamento abbiamo risolto il problema. L'algoritmo controlla tutte le regine e vede che nessuna è sotto attacco.  
Il primo spostamento poteva sembrare inutile, visto che il numero di attacchi subiti dalla prima regina non variava (sempre 1), ma ha iniziato un meccanismo di ricerca che ha portato alla soluzione finale.

## 9 Apprendimento automatico

Che cos'è l'apprendimento? non è la memorizzazione, bensì la comprensione di un processo. Compreso il meccanismo è possibile applicarlo anche su nuovi dati, questo processo è noto come **generalizzazione**.

La **classificazione** è un processo di indetificazione di un'osservazione ad una classe di appartenenza. I sistemi che sono in grado di dividere i vettori di caratteristiche in un numero finito di classi sono chiamati **classificatori**. In questo caso l'obiettivo nell'**apprendimento automatico** (**machine learning**) consiste nel generare una funzione dai dati raccolti e classificati che calcola il valore della classe per una nuova osservazione in base alle caratteristiche di quest'ultima. Nella figura seguente tale funzione è rappresentata dalla linea di demarcazione tracciata attraverso il diagramma.



Per  $n$  caratteristiche, il compito consiste nel trovare un **iperpiano**  $n - 1$  dimensionale all'interno dello spazio delle caratteristiche  $n$ -dimensionali che divide le classi nel miglior modo possibile.

Una divisione **buona** significa che la percentuale di oggetti classificati erroneamente è la più piccola possibile. Un **classificatore** mappa un vettore di caratteristiche su un valore di classe. La mappatura desiderata è anche chiamata **funzione target**

Se la funzione target non si mappa su un dominio finito, allora non è una classificazione, ma piuttosto un problema di **approssimazione**.

### 9.1 Machine Learning e Data Mining

Possiamo descrivere formalmente un **agente di apprendimento** come una funzione che mappa un vettore di caratteristiche su un valore di classe discreto o in generale su un numero reale.

La funzione nasce e si modifica durante la **fase di apprendimento**, influenzata dai dati di allenamento. Durante l'apprendimento, l'agente viene alimentato con i dati già classificati in precedenza da esperti umani.

**Definizione 1** L'apprendimento automatico (Machine Learning) è lo studio di algoritmi informatici che migliorano automaticamente attraverso l'esperienza. [Tom Mitchell]

**Definizione 2** Un agente è un agente di apprendimento se migliora nel tempo le sue prestazioni (misurate da un criterio appropriato) su dati nuovi e sconosciuti (dopo aver visto molti esempi di addestramento).

È importante testare la capacità di **generalizzazione** dell'algoritmo di apprendimento su dati sconosciuti, i **dati di test**, altrimenti ogni sistema sta attingendo alla propria memoria.

Un agente di apprendimento è caratterizzato dai seguenti termini:

- **Compito:** il compito dell'algoritmo di apprendimento è apprendere una mappatura.
- **Agente:** (più precisamente una classe di agenti) dobbiamo decidere quale algoritmo di apprendimento verrà utilizzato. Dalla scelta dell'agente si determina il **dominio** della funzione.
- **Dati di addestramento (esperienza):** : contengono la conoscenza che l'algoritmo di apprendimento dovrebbe estrarre e apprendere. È necessario assicurarsi che il campione sia informativo
- **Dati di test:** importanti per valutare se l'agente addestrato può generalizzare bene dai dati di addestramento a nuovi dati.
- **Misura delle prestazioni:** per testare la qualità di un agente

**Definizione:** Il processo di acquisizione della conoscenza dai dati, così come la loro rappresentazione e applicazione, è chiamato **data mining**. I metodi utilizzati sono generalmente presi dalla statistica o dall'apprendimento automatico e dovrebbero essere applicabili a quantità molto grandi di dati a costi ragionevoli.

## 9.2 Analisi dei dati

La statistica fornisce diversi modi per descrivere i dati con parametri semplici. Tra questi ne sceglieremo alcuni particolarmente importanti per l'analisi dei dati di allenamento e li testiamo su un insieme di dati medici raccolti in ambito ospedaliero.

Var. num.	Description	Values
1	Age	Continuous
2	Sex (1 = male, 2 = female)	1, 2
3	Pain quadrant 1	0, 1
4	Pain quadrant 2	0, 1
5	Pain quadrant 3	0, 1
6	Pain quadrant 4	0, 1
7	Local muscular guarding	0, 1
8	Generalized muscular guarding	0, 1
9	Rebound tenderness	0, 1
10	Pain on tapping	0, 1
11	Pain during rectal examination	0, 1
12	Axial temperature	Continuous
13	Rectal temperature	Continuous
14	Leukocytes	Continuous
15	Diabetes mellitus	0, 1
16	Appendicitis	0, 1

Il paziente numero uno  $x^1$  ad esempio, è descritto dal vettore

$$x^1 = (26, 1, 0, 0, 1, 0, \dots, 0, 1) \quad 16 \text{ valori}$$

Per ogni variabile (feature)  $x_i$ , la **media**  $|x_i|$  è definita come:

$$\bar{x}_i := \frac{1}{N} \cdot \sum_{p=1}^N x_i^p$$

e la **deviazione standard** si definisce come misura della sua deviazione media dal valore medio:

$$s_i := \sqrt{\frac{1}{N-1} \cdot \sum_{p=1}^N (x_i^p - \bar{x}_i)^2}$$

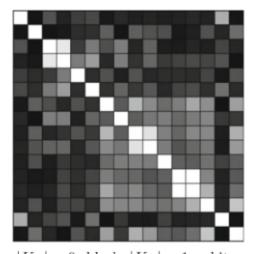
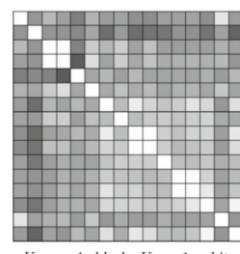
La **covarianza** definisce il grado di correlazione (statisticamente dipendenti) tra due variabili  $x_i$  e  $x_j$ . Se hanno segni diversi, il risultato è negativo (sensibilità di segno).

$$\sigma_{ij} = \frac{1}{N-1} \cdot \sum_{p=1}^N (x_i^p - \bar{x}_i)(x_j^p - \bar{x}_j)$$

il **coefficiente di correlazione** per due valori  $x_i$  e  $x_j$  non è che una covarianza normalizzata. La matrice  $K$  di tutti i coefficienti di correlazione contiene valori compresi  $[-1; 1]$  è **simmetrica** e tutti i suoi elementi diagonali hanno il valore 1

$$K_{ij} = \frac{\sigma_{ij}}{s_i \cdot s_j} \in [-1; 1]$$

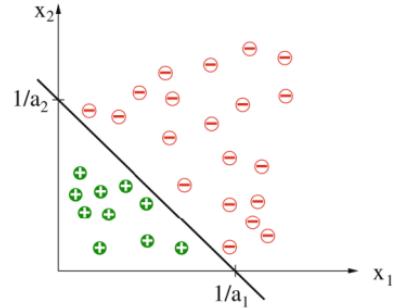
Questa matrice diventa un po' più leggibile quando la rappresentiamo come un **grafico di densità**. Invece dei valori numerici, gli elementi della matrice sono riempiti con valori grigi. Così possiamo vedere molto rapidamente quali variabili mostrano una dipendenza debole o forte



### 9.3 Il percettrone: un classificatore lineare

In questo caso i dati di addestramento sono separabili da una linea retta. Chiamiamo un tale insieme di dati di addestramento **linearmente separabili**. In  $n$  dimensioni è necessario un **iperpiano** per la separazione. Ogni iperpiano  $n - 1$  dimensionale in  $\mathbb{R}^n$  può essere descritto da un'equazione:

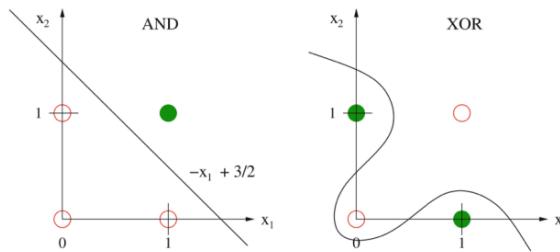
$$\sum_{i=1}^n a_i \cdot x_i = \theta$$



**Definizione** Due insiemi  $M_1$  e  $M_2$  (entrambi sottoinsiemi di  $\mathbb{R}^n$ ) sono detti **linearmente separabili** se esistono numeri reali  $a_1, \dots, a_n, \theta$  (dove  $\theta$  è chiamato il valore **soglia/threshold**) tali che:

$$\sum_{i=1}^n a_i \cdot x_i > \theta \quad \forall \mathbf{x} \in M_1 \quad \text{and} \quad \sum_{i=1}^n a_i \cdot x_i \leq \theta \quad \forall \mathbf{x} \in M_2$$

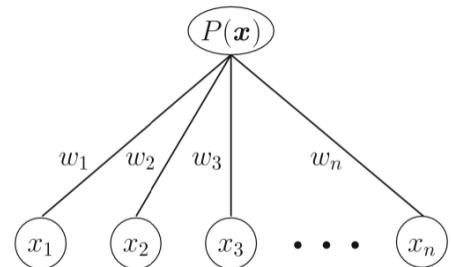
Seguono due esempi di funzioni, una linearmente separabile (la funzione logica AND) e una no (la funzione logica XOR).



**Il percettrone** è un algoritmo di apprendimento molto semplice che può separare insiemi separabili linearmente. Siano  $\mathbf{w} = (w_1, \dots, w_n) \in \mathbb{R}^n$  un vettore di pesi e  $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n$  un vettore di input. Un **percettrone** è una funzione  $P : \mathbb{R}^n \rightarrow \{0, 1\}$  che corrisponde alla seguente regola:

$$P(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^n w_i \cdot x_i > 0 \\ 0 & \text{else} \end{cases}$$

Il **percettrone** è un **algoritmo di classificazione** molto semplice. È equivalente a una **rete neurale** a due strati come in figura. Per ora, tuttavia, vedremo il percettrone solo come un agente di apprendimento, cioè come una funzione matematica che mappa un vettore di caratteristiche su un valore di funzione. Qui le variabili di input  $x_i$  sono denotate come **caratteristiche (features)**. L'iperpiano di separazione passa attraverso l'origine perché  $\theta = 0$ .



#### Regola d'apprendimento del percettrone

```

PERCEPTRONLEARNING[M+, M-]
w = arbitrary vector of real numbers
Repeat
  For all x ∈ M+
    If w · x ≤ 0 Then w = w + x
  For all x ∈ M-
    If w · x > 0 Then w = w - x
  Until all x ∈ M+ ∪ M- are correctly classified
  
```

Con la notazione  $M_+$  e  $M_-$  indichiamo gli insiemi di allenamento positivi e negativi. Il percettrone dovrebbe fornire il valore 1 per tutti gli  $x \in M_+$ , ossia quando  $\Rightarrow \mathbf{w} \cdot \mathbf{x} > 0$ . In caso contrario  $\mathbf{w} = \mathbf{w} + \mathbf{x}$  per cui il vettore peso viene cambiato esattamente nella giusta direzione (**ritaratura**). Lo vediamo quando applichiamo il percettrone al vettore modificato  $\mathbf{w} + \mathbf{x}$  perché:

$$(\mathbf{w} + \mathbf{x}) \cdot \mathbf{x} = \mathbf{w} \cdot \mathbf{x} + \mathbf{x}^2 > \mathbf{w} \cdot \mathbf{x}$$

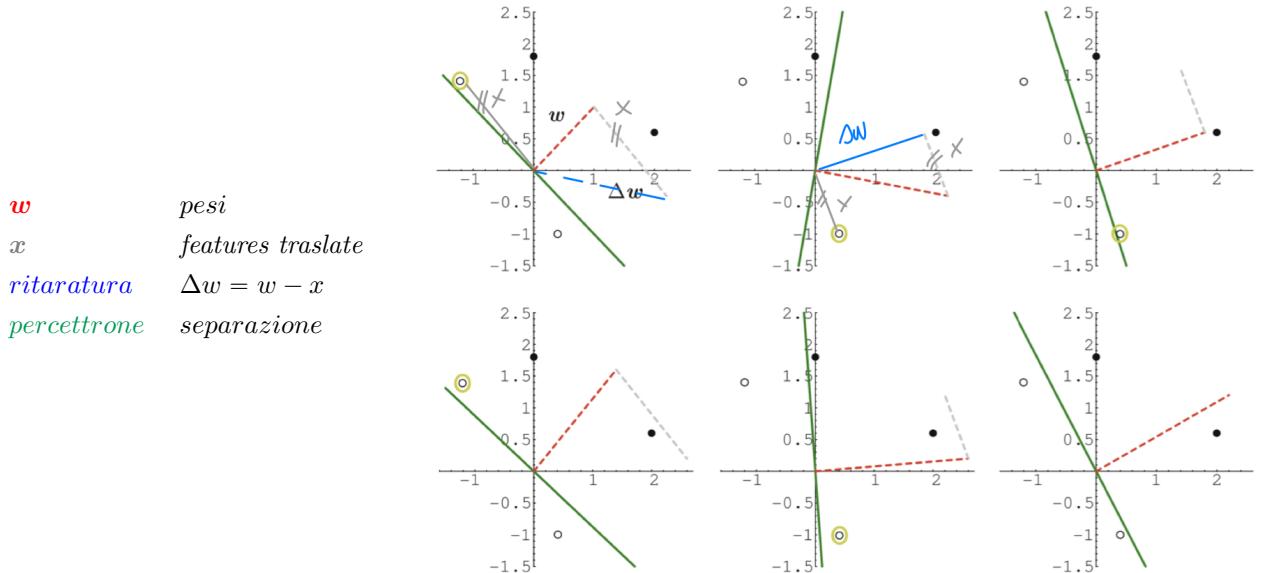
Se questo passaggio viene ripetuto abbastanza spesso, a un certo punto il valore  $\mathbf{w} \cdot \mathbf{x}$  diventerà **positivo**, come dovrebbe essere. Analogamente per i dati di addestramento negativi.

$$(\mathbf{w} - \mathbf{x}) \cdot \mathbf{x} = \mathbf{w} \cdot \mathbf{x} - \mathbf{x}^2 < \mathbf{w} \cdot \mathbf{x}$$

**Esempio:** un percettrone deve essere addestrato sugli insiemi  $M_+ = \{(0, 1.8), (2, 0.6)\}$  e  $M_- = \{(-1.2, 1.4), (0.4, -1)\}$ . Il vettore di peso iniziale  $\mathbf{w} = (1, 1)$ . Nella prima iterazione attraverso il ciclo dell'algoritmo di apprendimento, l'unico esempio di addestramento classificato erroneamente è  $(-1.2, 1.4)$  perché:

$$\begin{bmatrix} -1.2, & 1.4 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0.2 > 0$$

Ciò si traduce in  $\mathbf{w} = \mathbf{w} - \mathbf{x} = (1, 1) - (-1.2, 1.4) = (2.2, -0.4)$ . Dopo un totale di cinque modifiche, la linea di demarcazione si trova tra le due classi e il percettrone classifica quindi tutti i dati correttamente.



**Teorema:** Siano le classi  $M_+$  e  $M_-$  separabili linearmente da un iperpiano  $\mathbf{w}\mathbf{x} = 0$  ( $\theta = 0$  iperpiano passante per l'origine). Allora PERCEPTRONLEARNING converge con qualunque inizializzazione del vettore  $\mathbf{w}$ . Il percettrone  $P$  con il vettore peso così calcolato divide le classi  $M_+$  e  $M_-$ , ovvero:

$$P(x) = 1 \Leftrightarrow x \in M_+$$

$$P(x) = 0 \Leftrightarrow x \in M_-$$

Con il seguente trucco possiamo generare il termine costante. Poniamo  $x_n = 1 \in \mathbf{x}$ , e  $w_n = -\theta$ . Tale peso funziona come valore soglia perché:

$$\sum_{i=1}^n w_i \cdot x_i = \sum_{i=1}^{n-1} w_i \cdot x_i - \theta > 0 \Leftrightarrow \sum_{i=1}^{n-1} w_i \cdot x_i > \theta$$

Nell'applicazione dell'algoritmo di apprendimento, un bit con il valore costante 1 viene aggiunto al vettore dei dati di addestramento. Osserviamo che il **peso  $w_n$  (la soglia  $\theta$ ) viene appreso durante il processo di apprendimento.**

È stato dimostrato che un percettrone  $P_\theta : \mathbb{R}^{n-1} \rightarrow \{0, 1\}$  con una soglia arbitraria  $\theta$  può essere simulato da un percettrone  $P_\theta : \mathbb{R}^n \rightarrow \{0, 1\}$  con la soglia  $\theta = 0$ .

$$P_\theta(x_1, \dots, x_{n-1}) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n-1} w_i \cdot x_i > \theta \\ 0 & \text{else} \end{cases}$$

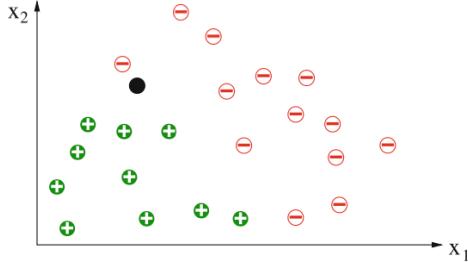
**Teorema:** Una funzione  $f : \mathbb{R}^n \rightarrow \{0, 1\}$  può essere rappresentata da un percettrone se e solo se i due insiemi di vettori di input positivi e negativi sono linearmente separabili.

## 9.4 Il metodo Nearest Neighbor (il vicino più vicino)

Per un percepitrone, la conoscenza disponibile nei dati di allenamento viene estratta e salvata in forma compressa nei pesi  $w_i$ . In tal modo le informazioni sui dati vengono perse. Ma questo è esattamente ciò che si desidera, se si suppone che il sistema faccia una generalizzazione dai dati di addestramento a nuovi dati. La **generalizzazione** in questo caso è un processo che richiede molto tempo e che ha l'obiettivo di trovare una funzione che classifichi i nuovi dati nel miglior modo possibile.

La memoria → memorizzazione in cui è anche possibile la generalizzazione ⇒ se si ha una buona percezione della **somiglianza** ⇒ applicare stessa soluzione utilizzata per il caso simile

**Cosa significa somiglianza nel contesto formale che stiamo costruendo?** Rappresentiamo i dati di addestramento nello spazio multidimensionale. **Definiamo che:** minore è la loro distanza nello spazio delle feature, più due osservazioni sono simili.



La distanza  $d(\mathbf{x}, \mathbf{y})$  tra due punti  $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$  e può essere misurata ad esempio dalla **distanza euclidea**

$$d(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Ci sono altre distanze che potrebbero essere utilizzate. La formula quindi diventa:

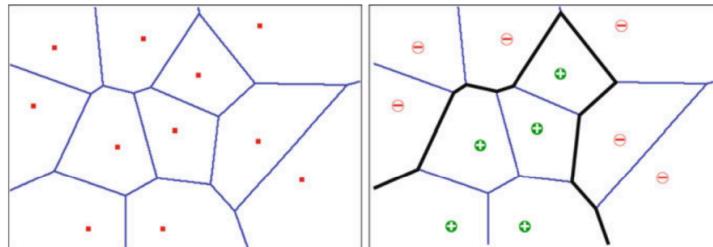
$$d_w(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^n w_i \cdot (x_i - y_i)^2}$$

Il seguente semplice programma di classificazione del vicino prossimo cerca tra i dati di addestramento il vicino  $t$  prossimo al nuovo esempio  $s$  e quindi classifica  $s$  esattamente come  $t$ .

```
NEARESTNEIGHBOR[M+, M-, s]
t = argmin_{x ∈ M+ ∪ M-} {d(s, x)}
If t ∈ M+ Then Return (,+)
Else Return (,-)
```

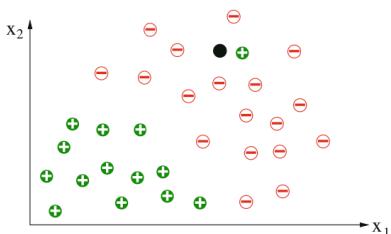
A differenza del percepitrone, il metodo del vicino prossimo non genera una linea che divide i dati di addestramento. Tuttavia, esiste certamente una **linea immaginaria** che separa le due classi. Possiamo generarla generando prima il cosiddetto **diagramma di Voronoi**.

Nel diagramma di Voronoi ogni punto è circondato da un poligono convesso. Dato un nuovo punto il NEARESTNEIGHBOR di quest'ultimo è quel punto nella cui zona il nuovo arrivato si trova.



**Possibile problema:** Un singolo punto classificato in maniera errata può portare in determinate circostanze a risultati di classificazione molto negativi. Un adattamento errato a errori casuali (rumore) è chiamato **overfitting**.

**NB:** fase di apprendimento si basa sulla memorizzazione per poi fare somiglianza



Per evitare false classificazioni dovute a singoli valori anomali, si consiglia di appianare, levigare in qualche modo la superficie di divisione. Introduzione **K-NEARESTNEIGHBOR**, che prende una decisione a maggioranza tra i  $k$  vicini più vicini.

## 9.5 La classificazione in più classi

La classificazione del vicino prossimo può essere applicata anche a più di due classi. Se il numero di classi è elevato, di solito non ha più senso utilizzare algoritmi di classificazione perché la dimensione dei dati di addestramento necessari cresce rapidamente con il numero di classi. Inoltre, in determinate circostanze, informazioni importanti vengono perse durante la classificazione.

Man mano che  $k$  diventa grande, tipicamente esistono più vicini con una grande distanza rispetto a quelli con una piccola distanza, in tal modo la mappatura è dominata dai vicini che sono lontani. Per evitare ciò, i  $k$  vicini sono ponderati in modo tale che i vicini più distanti abbiano un'influenza minore sul risultato, modificando i pesi come:

$$w_i = \frac{1}{1 + \alpha \cdot d(x, x_i)^2}$$

Ci sono molte alternative alla funzione peso (**chiamata anche kernel**), per la scelta di questi parametri liberi come per esempio  $\alpha$  vengono utilizzati dei metodi di ottimizzazione per l'impostazione automatica.

```
K-NEARESTNEIGHBOR( $M_+, M_-, s$ )
   $V = \{k \text{ nearest neighbors in } M_+ \cup M_-\}$ 
  If  $|M_+ \cap V| > |M_- \cap V|$  Then Return(,,+")
  ElseIf  $|M_+ \cap V| < |M_- \cap V|$  Then Return(,,-")
  Else Return(Random(,,+”,,,-”))
```

## 9.6 Confronto

```
PERCEPTRONLEARNING[ $M_+, M_-$ ]
 $w$  = arbitrary vector of real numbers
Repeat
  For all  $x \in M_+$ 
    If  $w \cdot x \leq 0$  Then  $w = w + x$ 
  For all  $x \in M_-$ 
    If  $w \cdot x > 0$  Then  $w = w - x$ 
  Until all  $x \in M_+ \cup M_-$  are correctly classified
```

```
NEARESTNEIGHBOR[ $M_+, M_-, s$ ]
 $t = \operatorname{argmin}_{x \in M_+ \cup M_-} \{d(s, x)\}$ 
If  $t \in M_+$  Then Return (,,+")
Else Return(,,-")
```

Confrontando il percepitrone con il metodo k-vicino-prossimo, ci accorgiamo che in quest'ultimo non accade nulla nella fase di apprendimento: nessun parametro che caratterizza l'algoritmo viene modificato.

Algoritmi di questo tipo sono anche denotati come **lazy learning** (apprendimento pigro) in contrasto con **eager learning** (apprendimento impaziente), in cui la fase di apprendimento può essere costosa, ma l'applicazione a nuovi esempi è molto efficiente.

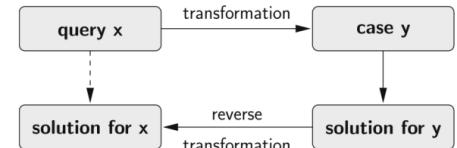
Il percepitrone e tutte le altre reti neurali, i decision tree e le reti Bayesiane sono metodi di apprendimento impaziente. Poiché i metodi di apprendimento pigro richiedono accesso alla memoria con tutti i dati di addestramento per approssimare un nuovo input, sono anche chiamati **apprendimento basato sulla memoria**.

## 9.7 Case Based Reasoning (CBR)

Nel ragionamento basato sui casi (CBR), il metodo del vicino prossimo viene esteso alle descrizioni dei **problem simbolici** e alle loro soluzioni. Data una query simbolica viene fornito un caso simile che deriva dal **case-base (database di casi)**, che corrisponde ai dati di addestramento nel metodo del k-vicino-prossimo.

Se prendessimo semplicemente il caso più simile, come facciamo nel metodo del k-vicino-prossimo, finiremmo per cercare di riparare la luce anteriore quando è la luce posteriore a essere rotta. Abbiamo quindi bisogno di una trasformazione della soluzione al problema trovato nel case-base simile alla query. I passaggi più importanti nella soluzione di un caso CBR sono

Feature	Query	Case from case base
Defective part:	Rear light	Front light
Bicycle model:	Marin Pine Mountain	VSF T400
Year:	1993	2001
Power source:	Battery	Dynamo
Bulb condition:	ok	ok
Light cable condition:	?	ok
	Solution	
Diagnosis:	?	Front electrical contact missing
Repair:	?	Establish front electrical contact



Le tre principali difficoltà per la costruzione di sistemi diagnostici CBR sono:

- **Modellazione:** i domini dell'applicazione devono essere modellati in un contesto formale. Lo sviluppatore può prevedere e mappare tutti i possibili casi speciali e le varianti di problemi?
- **Somiglianza:** trovare una metrica di somiglianza adatta per caratteristiche simboliche e non numeriche.
- **Trasformazione:** anche se viene trovato un caso simile, non è ancora chiaro che forma hanno la mappatura della trasformazione e la sua inversa

## 9.8 Decision Tree Learning

L'apprendimento con albero delle decisioni (Decision Tree Learning) è un algoritmo la cui conoscenza estratta non è solo disponibile e utilizzabile come una black box, ma può essere anche facilmente compresa, interpretata e controllata dagli esseri umani sotto forma di un albero decisionale leggibile.

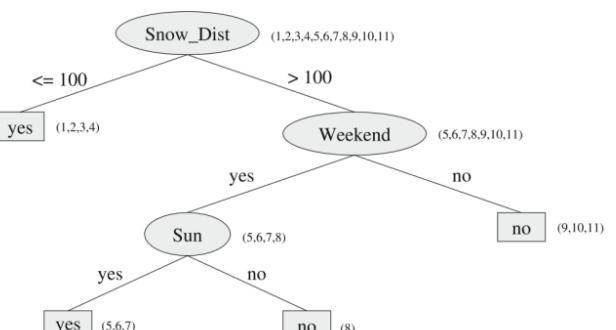
**NB:** Ciò rende l'apprendimento con albero decisionale uno strumento importante per il data mining.

**Esempio:** Uno sciatore appassionato vuole un albero decisionale che lo aiuti a decidere se vale la pena guidare la sua auto verso una stazione sciistica in montagna. Abbiamo quindi un problema a due classi: sì / no, basato sulle variabili elencate nella seguente tabella.

Variable	Value	Description
Ski (goal variable)	yes, no	Should I drive to the nearest ski resort with enough snow?
Sun (feature)	yes, no	Is there sunshine today?
Snow_Dist (feature)	$\leq 100$ , $> 100$	Distance to the nearest ski resort with good snow conditions (over/under 100 km)
Weekend (feature)	yes, no	Is it the weekend today?

Un **albero decisionale** è un albero i cui:

- **nodi interni** rappresentano gli attributi del problema
- i **rami** rappresentano i diversi valori di questi attributi
- **nodi foglia** rappresentano i valori della classificazione.



I dati utilizzati per la costruzione dell'albero decisionale sono mostrati nella seguente tabella.

La riga 6 e la riga 7 si contraddicono a vicenda. Pertanto, nessun algoritmo di classificazione deterministico può classificare correttamente tutti i dati. Il numero di dati classificati erroneamente deve quindi essere  $\geq 1$ . **Come viene creato un albero del genere dai dati?** Un semplice, ovvio algoritmo per la costruzione di un albero genererebbe semplicemente tutti gli alberi, quindi per ogni albero calcolare il numero di classificazioni errate dei dati, e alla fine scegliere l'albero con il numero minimo di errori. L'ovvio svantaggio di questo algoritmo è il suo **tempo di calcolo inaccettabilmente alto**, non appena il numero di attributi cresce.

Day	Snow_Dist	Weekend	Sun	Skiing
1	$\leq 100$	yes	yes	yes
2	$\leq 100$	yes	yes	yes
3	$\leq 100$	yes	no	yes
4	$\leq 100$	no	yes	yes
5	$> 100$	yes	yes	yes
6	$> 100$	yes	yes	yes
7	$> 100$	yes	yes	no
8	$> 100$	yes	no	no
9	$> 100$	no	yes	no
10	$> 100$	no	yes	no
11	$> 100$	no	no	no

Sviluppiamo un **algoritmo euristico** che, partendo dalla radice, costruisce ricorsivamente un albero decisionale:

- viene scelto l'attributo con il maggior guadagno di informazione (information gain) (Snow\_Dist) per il nodo radice dall'insieme di tutti gli attributi.
- Per ogni valore di attributo ( $\leq 100$ ,  $> 100$ ) c'è un ramo nell'albero.
- Ora per ogni ramo questo processo viene ripetuto ricorsivamente: scelta attributo non ancora utilizzato con maggior information gain per la creazione del nodo

## 9.9 Entropia come misura di informazione

Introduciamo ora l'entropia come metrica per il contenuto informativo di un insieme di dati di allenamento  $D$ . Se guardiamo solo la variabile binaria skiing nell'esempio sopra, allora  $D$  può essere descritto come

$$D = (\text{yes}, \text{yes}, \text{yes}, \text{yes}, \text{yes}, \text{yes}, \text{no}, \text{no}, \text{no}, \text{no}, \text{no})$$

con probabilità stimate

$$p_1 = P(\text{yes}) = \frac{6}{11} \quad \text{and} \quad p_2 = P(\text{no}) = \frac{5}{11}$$

La una distribuzione di probabilità è  $\mathbf{p} = (6/11, 5/11)$ . In generale, per un problema con  $n$  classi si ha:

$$\mathbf{p} = (p_1, \dots, p_n) \quad \text{con} \quad \sum_{i=1}^n p_i = 1$$

**Esempi limite:**

$$\mathbf{p} = (1, 0, \dots, 0)$$

Una distribuzione di probabilità in cui il primo degli  $n$  eventi accadrà sicuramente e tutti gli altri no. L'incertezza sull'esito degli eventi è quindi minima. **Quanti bit sarebbero stati necessari per codificare un evento del genere? [Claude Shannon]** sono necessari 0 bit, perché sappiamo che il caso  $i = 1$  si verifica sempre.

$$\mathbf{p} = \left( \frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n} \right)$$

l'incertezza è massima perché nessun evento può essere distinto dagli altri. **Quanti bit sarebbero stati necessari per codificare un evento del genere? [Claude Shannon]**  $n$  possibilità ugualmente probabili, quindi per la codifica binaria sono necessari  $\log_2(1/p_i)$  bit.

Nel caso generale  $\mathbf{p} = (p_1, \dots, p_n)$  se le probabilità degli eventi elementari deviano dalla distribuzione uniforme, allora bisogna calcolare un **valore atteso H** per il numero di bit necessari. A tal fine peseremo tutti i valori dei bit necessari per ciascun evento  $\log_2(1/p_i) = -\log_2(p_i)$  con la probabilità dell'evento stesso e otteniamo:

$$H = \sum_{i=1}^n p_i(-\log_2(p_i)) = -\sum_{i=1}^n p_i \cdot \log_2(p_i)$$

Maggiore è l'incertezza sul risultato, maggiore è il numero di bit di cui abbiamo bisogno per codificare un evento. **L'entropia H** viene quindi definita come metrica per l'incertezza di una distribuzione di probabilità

$$H(\mathbf{p}) = H(p_1, \dots, p_n) := -\sum_{i=1}^n p_i \cdot \log_2(p_i)$$

Poiché a ciascun set di dati  $D$  classificato viene assegnata una distribuzione di probabilità  $\mathbf{p}$  stimando le probabilità di ciascuna classe

$$H(D) = H(\mathbf{p})$$

Ora, poiché il **contenuto informativo**  $I(D)$  dell'insieme di dati  $D$  è inteso essere l'opposto dell'incertezza. Quindi definiamo il contenuto informativo di un set di dati  $D$  come:

$$I(D) := 1 - H(D)$$

Se applichiamo la formula dell'entropia all'esempio dello sciatore, il risultato è

$$H(D) = H(6/11, 5/11) = 0.994$$

Durante la costruzione di un albero decisionale, il set di dati viene ulteriormente suddiviso per ogni nuovo attributo. Più un attributo aumenta il contenuto informativo della distribuzione dividendo i dati, migliore è l'attributo. Il **guadagno di informazione** (**information gain**)  $G(D, A)$  attraverso l'uso dell'attributo  $A$  è determinato:

- dalla differenza tra il contenuto informativo medio del dataset  $D = D_1 \cup \dots \cup D_n$  diviso per l'attributo a  $n$  valori  $A$
- e il contenuto informativo  $I(D)$  dell'insieme di dati non diviso

$$\begin{aligned} G(D, A) &= \sum_{i=1}^n \frac{|D_i|}{|D|} \cdot I(D_i) - I(D) \\ &= \sum_{i=1}^n \frac{|D_i|}{|D|} \cdot \left(1 - H(D_i)\right) - \left(1 - H(D)\right) \\ &= 1 - \sum_{i=1}^n \frac{|D_i|}{|D|} \cdot H(D_i) - 1 + H(D) \\ &= H(D) - \sum_{i=1}^n \frac{|D_i|}{|D|} \cdot H(D_i) \end{aligned}$$

Applicato al nostro esempio:

$$\begin{aligned} G(D, \text{Snow\_Dist}) &= H(D) - \left( \frac{4}{11} H(D)_{\leq 100} + \frac{7}{11} H(D)_{> 100} \right) \\ &= 0.994 - \left( \frac{4}{11} \cdot 0 + \frac{7}{11} \cdot 0.863 \right) = 0.445 \end{aligned}$$

$$G(D, \text{Weekend}) = 0.150 \quad G(D, \text{Sun}) = 0.049$$

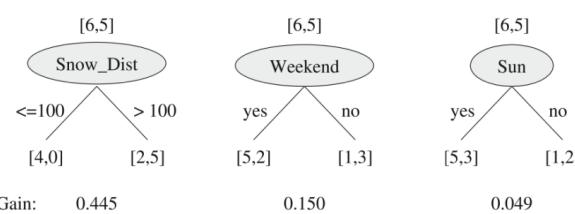
Day	Snow_Dist	Weekend	Sun	Skiing
1	$\leq 100$	yes	yes	yes
2	$\leq 100$	yes	yes	yes
3	$\leq 100$	yes	no	yes
4	$\leq 100$	no	yes	yes
5	$> 100$	yes	yes	yes
6	$> 100$	yes	yes	yes
7	$> 100$	yes	yes	no
8	$> 100$	yes	no	no
9	$> 100$	no	yes	no
10	$> 100$	no	yes	no
11	$> 100$	no	no	no

$H(D \leq 100)$   
Lo skiing è sempre yes entropia = 0  
(certezza totale)

$H(D > 100)$

L'attributo **Snow\_Dist** quindi diventa il nodo radice dell'albero decisionale. Questo perché ha il guadagno informativo più alto = ossia, dividendo i dati sulla base di questo attributo abbiamo il maggior sbilanciamento tra andare a sciare e non.

[conteggio dei si, conteggio dei no]



I due valori di attributo  $\leq 100$  e  $> 100$  generano due archi nell'albero, che corrispondono ai sottoinsiemi  $D_{\leq 100}$  e  $D_{> 100}$ . Per il sottoinsieme  $D_{\leq 100}$  la classificazione è chiaramente **si** quindi l'albero termina qui. Nell'altro ramo  $D_{> 100}$  non c'è un risultato chiaro, quindi l'algoritmo si ripete ricorsivamente.

$$G(D_{> 100}, \text{Weekend}) = 0.292 \quad G(D_{> 100}, \text{Sun}) = 0.170$$

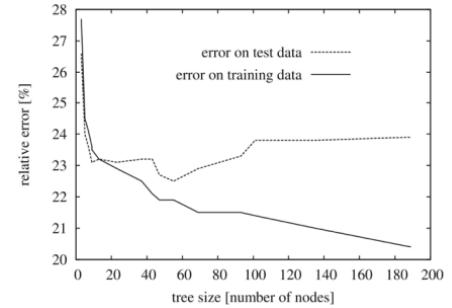
Scegliamo quello con **information gain** maggiore ossia  $G(D_{> 100}, \text{Weekend})$ . Al nodo si assegna perciò l'attributo **Weekend**. Per **Weekend = no** l'albero termina con la decisione **Ski = no**.  $G(D_{\text{weekend}}, \text{Sun}) = 0.171$

## 9.10 Potatura dell'albero

**Rasoio di Occam:** tra due teorie scientifiche che spiegano ugualmente bene la stessa situazione, si preferisce quella **più semplice**. Oltre a una migliore comprensibilità il rasoio di Occam ha la capacità di **generalizzazione**. Più complesso è il modello (in questo contesto un albero decisionale), più dettagli sono rappresentati, ma nella stessa misura meno il modello è trasferibile a nuovi dati.

L'obiettivo di ogni algoritmo per la generazione di un albero decisionale deve essere quello di generare l'albero decisionale più piccolo possibile (più facilmente comprensibile) per un dato tasso di errore. È importante che l'albero appreso non solo memorizzi i dati di allenamento, ma che generalizzi bene. Per testare la capacità di generalizzare di un albero, dividiamo in training set e testing set.

Il tasso di errore sui dati di addestramento diminuisce con l'aumentare della dimensione dell'albero. Fino a una dimensione dell'albero di 55 nodi, diminuisce anche il tasso di errore sui dati di test. Se l'albero cresce ulteriormente, tuttavia, il tasso di errore ricomincia ad aumentare → **overfitting**



Come possiamo ora trovare questo punto di minimo errore sui dati del test? L'algoritmo più ovvio è chiamato **convalida incrociata (cross-validation)**. Durante la costruzione dell'albero, l'errore sui dati di test viene misurato in parallelo. Non appena l'errore aumenta in modo significativo, viene salvato l'albero con l'errore minimo.

**Dataset con dati mancanti:** tali dati possono tuttavia essere utilizzati durante la costruzione dell'albero decisionale. Possiamo assegnare all'attributo:

- o il valore più frequente dell'intero set di dati
- il più frequente di tutti i punti dati della stessa classe.

## 9.11 Cross-validation

Molti algoritmi di apprendimento hanno il problema dell'eccessivo adattamento (overfitting), ossia adattare la complessità del modello appreso alla complessità dei dati di addestramento e quindi inseguire il rumore presente nei dati.

Con la cross-validation si tenta di ottimizzare la complessità del modello in modo tale da ridurre al minimo l'errore di classificazione o di approssimazione su un set di dati di test finora mai visti. Ciò richiede che la complessità del modello sia controllabile da un parametro  $\gamma$ .

Variamo il parametro  $\gamma$  durante l'addestramento dell'algoritmo su un set di dati di addestramento e scegliamo il valore di  $\gamma$  che riduce al minimo l'errore su un set di dati di test indipendente.

L'intero set di dati  $X$  è diviso in  $k$  blocchi di uguali dimensioni. Quindi l'algoritmo viene addestrato  $k$  volte su  $k - 1$  blocchi e testato sul blocco rimanente. I  $k$  errori calcolati vengono mediati e il valore  $\gamma_{opt}$  con l'errore medio più piccolo viene scelto per addestrare il modello finale sull'intero set di dati  $X$ .

```

CROSSVALIDATION( $\mathbf{X}, k$ )
Partition data into  $k$  equally sized blocks  $\mathbf{X} = \mathbf{X}_1 \cup \dots \cup \mathbf{X}_k$ 
For all  $\gamma \in \{\gamma_{min}, \dots, \gamma_{max}\}$ 
  For all  $i \in \{1, \dots, k\}$ 
    Train a model of complexity  $\gamma$  on  $\mathbf{X} \setminus \mathbf{X}_i$ 
    Compute the error  $E(\gamma, \mathbf{X}_i)$  on the test set  $\mathbf{X}_i$ 
  Compute the mean error  $E(\gamma) = \frac{1}{k} \sum_{i=1}^k E(\gamma, \mathbf{X}_i)$ 
  Choose the value  $\gamma_{opt} = \operatorname{argmin}_{\gamma} E(\gamma)$  with smallest mean error
  Train the final model with complexity  $\gamma_{opt}$  on the whole data set  $\mathbf{X}$ 

```

### Overfitting - bias variance tradeoff:

- modello eccessivamente semplice → l'approssimazione non ottimale dei dati → bias ↑ var ↓
- modello eccessivamente complesso → overfitting dati addestramento → bias ↓ var ↑

## 9.12 One-class Learning

Le attività di classificazione nell’apprendimento supervisionato richiedono che a tutti i dati di formazione siano assegnate etichette di classe. Tuttavia, ci sono applicazioni per le quali è disponibile solo una classe di etichette. Pertanto è necessario un algoritmo di apprendimento che può, sulla base dei dati privi di errori, riesce a classificare le osservazioni che appartengono o meno a tale classe. A tale scopo, esiste tra gli altri un algoritmo chiamato **Nearest Neighbor Data Description (NNDD)**.

## 9.13 Nearest Neighbor Data Description

NNDD appartiene alla categoria degli algoritmi di apprendimento pigro, durante l’apprendimento le uniche cose che accadono sono la normalizzazione e la memorizzazione dei vettori delle features. La **normalizzazione** di ogni singola caratteristica è necessaria per dare a ciascuna caratteristica lo stesso peso e per non avere problemi con il rumore.

Sia  $\mathbf{X} = (\mathbf{x}_1; \dots; \mathbf{x}_n)$  un insieme di addestramento costituito da  $n$  vettori di caratteristiche (features). Un nuovo punto  $\mathbf{q}$  che deve ancora essere classificato è accettato se la distanza dal vicino prossimo non è maggiore di  $\gamma\bar{D}$ :

$$D(\mathbf{q}, NN(\mathbf{q})) \leq \gamma\bar{D}$$

$D(x, y)$  è una metrica della distanza, ad esempio, qui è usata la distanza euclidea. La funzione restituisce il vicino prossimo a  $\mathbf{q}$

$$NN(\mathbf{q}) = \arg \min_{\mathbf{x} \in \mathbf{X}} \{D(\mathbf{q}, \mathbf{x})\}$$

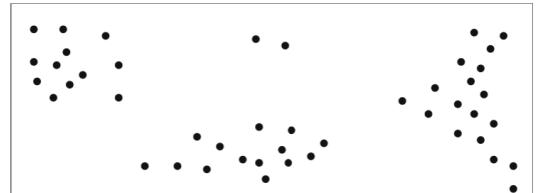
e  $\bar{D}$  è la distanza media dei vicini più vicini a tutti i punti dati in  $\mathbf{X}$ . Per calcolare  $\bar{D}$  viene calcolata la distanza di ciascun punto dal suo vicino prossimo, dunque  $\bar{D}$  è la media aritmetica di queste distanze calcolate. Determinare  $\gamma$  tramite cross-validation.

$$\bar{D} = \frac{1}{n} \cdot \sum_{i=1}^n D(\mathbf{x}_i, NN(\mathbf{x}_i))$$

## 9.14 Clustering

La distinzione del clustering (raggruppamento) dall’apprendimento supervisionato è che i dati di addestramento non sono etichettati. Trovare le strutture è il punto centrale del clustering.

In un cluster, la distanza dei punti adiacenti è in genere inferiore alla distanza tra i punti di cluster diversi. Pertanto la scelta di una **metrica di distanza** adeguata per i punti, cioè per gli oggetti da raggruppare e per i cluster, è di fondamentale importanza



**Metriche della distanza:** per ciascuna applicazione le metriche sono definite per la distanza  $d$  tra due vettori  $x, y \in \mathbb{R}^n$ :

*distanza euclidea*

$$d_e(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

*somma delle distanze al quadrato*

$$d_q(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n (x_i - y_i)^2$$

*distanza di Manhattan*

$$d_m(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n |x_i - y_i|$$

*distanza del componente massimo*

$$d_\infty(\mathbf{x}, \mathbf{y}) = \max_{i=1, \dots, n} |x_i - y_i|$$

*inverso somiglianza di 2 vettori*

$$d_s(\mathbf{x}, \mathbf{y}) = \frac{|\mathbf{x}| \cdot |\mathbf{y}|}{\mathbf{x} \cdot \mathbf{y}}$$

Nel contesto della classificazione dei testi, si usa spesso il prodotto scalare normalizzato poiché è una metrica per la somiglianza dei due vettori. Come metrica della distanza si usa spesso l’inverso

$$\frac{\mathbf{x} \cdot \mathbf{y}}{|\mathbf{x}| |\mathbf{y}|}$$

## 9.15 k-Means e l'algoritmo EM

Ogni volta che il numero di cluster è già noto in anticipo, è possibile utilizzare l'algoritmo **k-Means**. Come suggerisce il nome,  $k$  cluster sono definiti dal loro valore medio. I  $k$  punti medi dei cluster  $\mu_1, \dots, \mu_k$  vengono inizializzati alle coordinate di  $k$  punti selezionati casualmente o manualmente. Quindi i due passaggi seguenti vengono eseguiti ripetutamente:

- classificazione di tutti i dati rispetto al punto medio del cluster più vicino
- ricalcolo del punto medio del cluster sulla base dei punti classificati precedentemente per una stima più corretta del punto medio del cluster

```

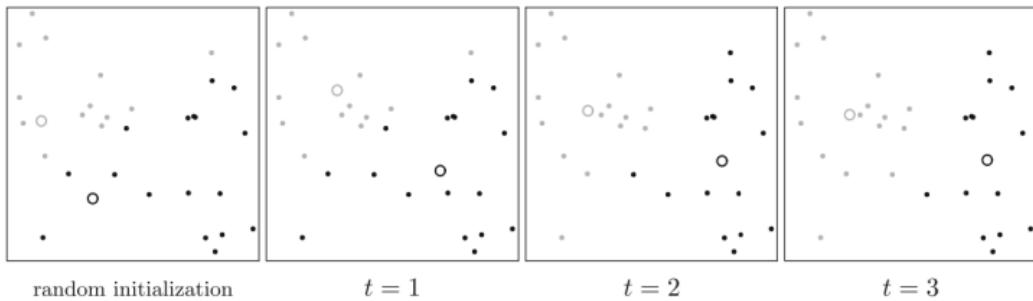
K-MEANS( $x_1, \dots, x_n, k$ )
initialize cluster centers  $\mu_1 = x_{i_1}, \dots, \mu_k = x_{i_k}$  (e.g. randomly)
Repeat
    classify  $x_1, \dots, x_n$  to each's nearest  $\mu_i$ 
    recalculate  $\mu_1, \dots, \mu_k$ 
Until no change in  $\mu_1, \dots, \mu_k$ 
Return( $\mu_1, \dots, \mu_k$ )

```

Il calcolo del punto medio del cluster  $\mu$  per i punti  $x_1, \dots, x_l$  viene eseguito così:

$$\mu = \frac{1}{l} \cdot \sum_{i=1}^l x_i$$

**Esempio:** k-means con due classi ( $k = 2$ ) applicate a 30 punti dati. All'estrema sinistra il set di dati con i centri iniziali e a destra i cluster dopo ogni iterazione. Dopo tre iterazioni si raggiunge la convergenza.



Osservazioni esempio:

- Vediamo come, dopo tre iterazioni, **i centri di classe, che sono stati inizialmente scelti a caso, si stabilizzano**. I punti vengono classificati sulla base della loro distanza da questi centroidi in grigio o nero. Successivamente vengono calcolati i nuovi centroidi come media vettoriale di tutti i punti neri o grigi e viene eseguita una nuova riclassificazione più precisa di prima
- Sebbene questo algoritmo non garantisca la convergenza, di solito **converge molto rapidamente**. Ciò significa che il numero di passaggi di iterazione è in genere molto inferiore al numero di punti dati.
- La sua complessità è  $O(n \cdot d \cdot k \cdot t)$ :  $n$  numero totale di punti,  $d$  la dimensionalità dello spazio delle caratteristiche,  $t$  il numero di passaggi di iterazione.

**L'algoritmo EM:** è una variante continua di k-means, poiché non effettua un'assegnazione definitiva dei dati alle classi, piuttosto, **per ogni punto restituisce la probabilità che appartenga alle varie classi**.

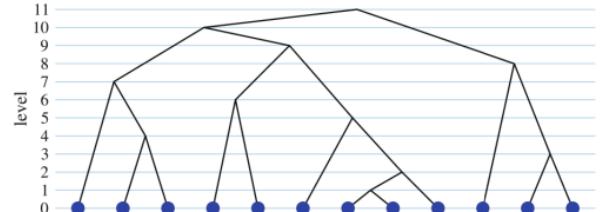
Qui dobbiamo presumere che il tipo di distribuzione di probabilità sia noto. Spesso viene utilizzata la distribuzione normale e si calcola i parametri statistici più importanti. I passaggi sono:

- **aspettativa (Expectation):** per ogni punto dati viene calcolata la probabilità  $P(C_j|x_i)$  che appartenga a ciascun cluster;
- **massimizzazione (Maximization):** utilizzando le probabilità appena calcolate, i parametri della distribuzione vengono ricalcolati.

## 9.16 Clustering gerarchico

Inizialmente ogni punto é un cluster, quindi dati  $n$  punti avremo  $n$  cluster. I cluster piú vicini (punti piú vicini) tra loro vengono combinati fino a quando tutti i punti sono stati combinati in un unico cluster, o fino a quando non viene raggiunto un criterio di terminazione.

```
HIERARCHICALCLUSTERING( $x_1, \dots, x_n, k$ )
initialize  $C_1 = \{x_1\}, \dots, C_n = \{x_n\}$ 
Repeat
    Find two clusters  $C_i$  and  $C_j$  with the smallest distance
    Combine  $C_i$  and  $C_j$ 
Until Termination condition reached
Return(tree with clusters)
```

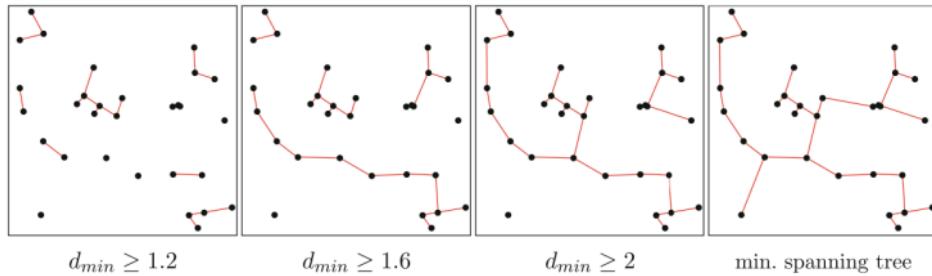


**La condizione di terminazione:** potrebbe essere, ad esempio, un numero desiderato di cluster viene raggiunto, oppure una distanza massima tra i cluster viene raggiunta.

La metrica per il calcolo delle distanze tra cluster sono diverse da quelle precedentemente definite. Una metrica comoda e spesso utilizzata dati due cluster  $C_i$  e  $C_j$  sono:

$$\begin{aligned} \text{la distanza tra i due punti più vicini: } & d_{min}(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y) \\ \text{la distanza tra i due punti più lontani: } & d_{max}(C_i, C_j) = \max_{x \in C_i, y \in C_j} d(x, y) \\ \text{la distanza del punto medio del cluster: } & d_\mu(C_i, C_j) = d(\mu_i, \mu_j) \end{aligned}$$

**Otteniamo così l'algoritmo del vicino prossimo:** L'algoritmo del vicino piú vicino applicato ai dati che avevamo prima diviso in 2 cluster con k-means, a diversi livelli con 12, 6, 3, 1 cluster.



L'algoritmo genera uno **spanning tree minimo**, ossia un grafo aciclico e non orientato con la somma minima delle lunghezze dei lati, dove il risultato dipende fortemente dall'algoritmo o dalla metrica di distanza scelta.

Per un'implementazione efficiente di questo algoritmo, creiamo prima una matrice di adiacenza in cui vengono salvate le distanze tra tutti i punti, che richiede tempo e memoria  $O(n^2)$ . Se il numero di cluster non ha un limite superiore, il ciclo itererà  $n - 1$  volte e il tempo di calcolo asintotico diventa  $O(n^3)$ .

## 9.17 Come determinare il numero di cluster?

- In tutti gli algoritmi di clustering discussi finora, l'utente deve specificare il numero di cluster e solitamente non si sa quale sia un numero appropriato per avere un buon partizionamento.
- Un metodo euristico per valutare un raggruppamento utilizzando il **criterio della larghezza della sagoma (silhouette width, SW)**. Il calcolo combinatorio di tutte le possibili scelte di partizionamento dei  $n$  dati e applicazione successiva di SW non é un metodo pratico. Possiamo utilizzare k-means lasciarlo funzionare per  $k$  da 1 a  $n$  e utilizzare il criterio SW per determinare il  $k$  migliore e la sua rispettiva partizione
- **Criterio che misuri la qualità di una partizione:** la distanza media tra due punti arbitrari all'interno dello stesso cluster dovrebbe essere inferiore alla distanza tra due punti arbitrari che si trovano in diversi cluster vicini. Il rapporto tra la distanza media tra i punti nei cluster vicini e la distanza media tra i punti all'interno del cluster dovrebbe essere massimizzato.

Dati i punti  $(\mathbf{x}_1, \dots, \mathbf{x}_n)$  e un funzione di raggruppamento  $c$  che assegna ogni punto a un cluster.

$$c : \mathbf{x}_i \rightarrow c(\mathbf{x}_i)$$

Sia  $\bar{d}(i, l)$  la distanza media da  $\mathbf{x}_i$  a tutti i punti che non sono  $\mathbf{x}_i$  nel cluster  $l$ . Allora  $a(i) = \bar{d}(i, c(\mathbf{x}_i))$  è la distanza media da  $\mathbf{x}_i$  a tutti gli altri punti nel proprio cluster. Se  $\mathbf{x}_i$  è l'unico punto nel cluster poniamo  $a(i) = 0$ .

$$b(i) = \min_{j \neq c(\mathbf{x}_i)} \{\bar{d}(i, j)\}$$

$b(i)$  è la distanza media più piccola dal punto  $\mathbf{x}_i$  a un cluster a cui  $\mathbf{x}_i$  non appartiene. Definiamo allora la seguente funzione: che misura quanto bene il punto  $\mathbf{x}_i$  sta in un cluster.

$$s(i) = \begin{cases} 0 & \text{if } a(i) = 0 \\ \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} & \text{otherwise} \end{cases}$$

I valori possibili di  $s(i)$  sono:

- -1 se  $\mathbf{x}_i$  è nel cluster sbagliato
- 0 se  $\mathbf{x}_i$  è tra due cluster
- 1 se  $\mathbf{x}_i$  è nel mezzo di un cluster ben delineato

**Criterio SW:** cerchiamo una partizione che massimizzi il valor medio di  $s(i)$  su tutti i punti:

$$S = \frac{1}{n} \cdot \sum_{i=1}^n s(i)$$

**Lavorando con il clustering k-means, questa massimizzazione può essere fatto con l'algoritmo OMRk (Ordered Multiple Runs k):** Questo algoritmo applica ripetutamente k-means per diversi valori di  $k$ . Poiché il risultato di k-means dipende fortemente dalla sua inizializzazione, per ogni  $k$ , vengono provate  $p$  diverse inizializzazioni casuali nel ciclo interno, e quindi la funzione restituisce l'ottimo  $k^*$  e la corrispondente partizione migliore  $P^*$

```

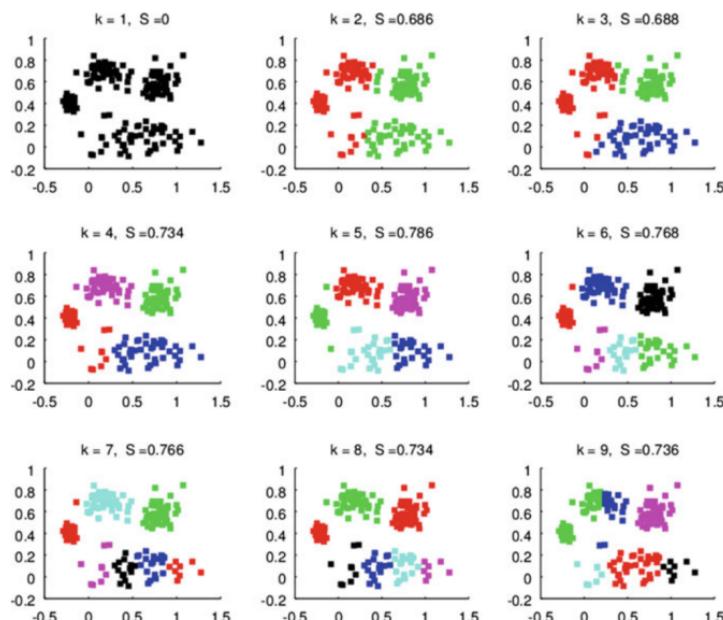
OMRK( $\mathbf{x}_1, \dots, \mathbf{x}_n, p, k_{max}$ )
 $S^* = -\infty$ 
For  $k = 2$  To  $k_{max}$ 
  For  $i=1$  To  $p$ 
    Generate a random partition with  $k$  clusters (initialization)
    Obtain a partition  $P$  with k-means
    Determine  $S$  for  $P$ 
    If  $S > S^*$  Then
       $S^* = S; k^* = k; P^* = P$ 
Return  $(k^*, P^*)$ 

```

L'algoritmo OMRk può essere utilizzato anche con altri algoritmi di clustering come l'algoritmo EM e il clustering gerarchico

**Esempio:** risultati dell'algoritmo OMRk per  $k$  = da 2 a 9.

Il miglior valore di  $S = 0,786$  è stato trovato con  $k = 5$ .



## 10 Reti Neurali

- Partendo dalla conoscenza della funzione delle reti neurali naturali, tentiamo di modellarle, simularle e persino ricostruirle sotto forma di hardware.
- Dalla prospettiva attuale, non vale più nemmeno la pena di provare a fare uno schema circuitale completo del cervello, perché la struttura del cervello è adattativa. Cambia e si adatta in base alle attività dell'individuo e alle influenze ambientali.
- Neurone ha un assone, che può stabilire connessioni locali con altri neuroni sui dendriti. Il corpo cellulare del neurone può immagazzinare piccole cariche elettriche, se la tensione supera una certa soglia, il neurone si attiverà.

### 10.1 Il modello matematico

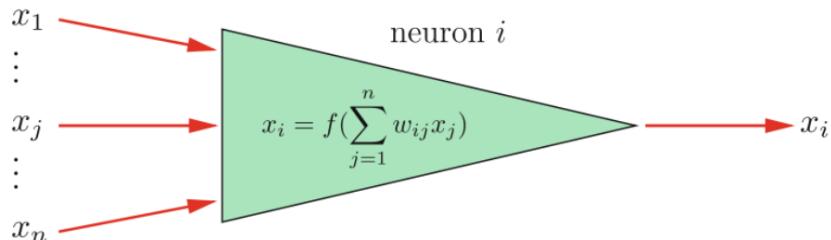
Sostituiamo l'asse temporale continuo con una scala temporale discreta. Il neurone  $i$  esegue il seguente calcolo in una fase temporale. Il “caricamento” del potenziale di attivazione **si ottiene semplicemente sommando** i valori di uscita ponderati  $x_1, \dots, x_n$  di tutte le connessioni in ingresso nella formula

$$\sum_{j=1}^n w_{ij} \cdot x_j$$

Quindi viene applicata una **funzione di attivazione**  $f$  e il risultato viene trasmesso ai neuroni vicini come output.

$$x_i = f\left(\sum_{j=1}^n w_{ij} \cdot x_j\right)$$

La struttura di un neurone formale, che applica la funzione di attivazione  $f$  alla somma ponderata di tutti gli input



Scelta funzione di attivazione  $f$ :

- L'identità**  $f(x) = x$ . Il neurone calcola quindi solo la somma ponderata dei valori di input e la trasmette. Tuttavia, questo spesso porta a **problemi di convergenza** con la dinamica neurale perché la funzione  $f(x) = x$  è illimitata e i valori della funzione possono crescere nel tempo a dismisura.
- Ben limitata, al contrario, è la **funzione di soglia**: dove la soglia viene identificata da  $\Theta$ .

$$H_\Theta(x) = \begin{cases} 0 & \text{if } x < \Theta \\ 1 & \text{else.} \end{cases}$$

L'intero neurone allora calcola il suo output così:

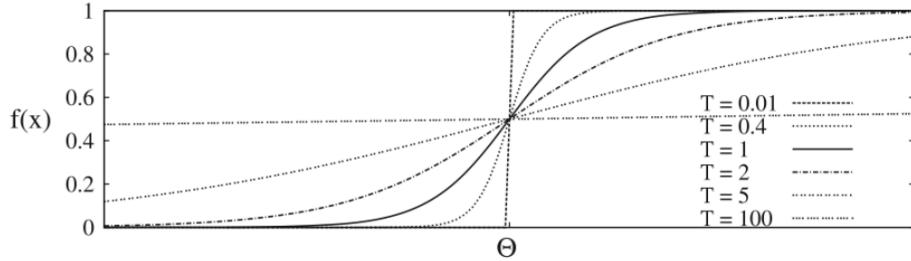
$$x_i = \begin{cases} 0 & \text{if } \sum_{j=1}^n w_{ij} \cdot x_j < \Theta \\ 1 & \text{else.} \end{cases}$$

che è identico a come si comporta un percepitrone con soglia  $\Theta$ . La funzione soglia ha senso per i **neuroni binari** perché l'attivazione di un neurone può assumere comunque solo i valori 0 o 1. Al contrario, per **neuroni continui** con attivazioni comprese tra  $[0, 1]$ . Tuttavia, questo può essere attenuato da una **funzione sigmoide**

- Funzione sigmoide**: per neuroni continui. In prossimità dell'area critica intorno alla soglia  $\Theta$ , questa funzione si comporta quasi in modo lineare e ha un limite asintotico.

$$f(x) = \frac{1}{1 + e^{-\frac{x-\Theta}{T}}}$$

La **smoothness** può essere variata dal parametro  $T$ , come mostrato nella seguente figura.



La modellazione dell'apprendimento è fondamentale per la teoria delle reti neurali. Una possibilità di apprendimento consiste nel rafforzare una sinapsi in base a quanti impulsi elettrici deve trasmettere. Questo principio è stato postulato da Donald Hebb nel 1949 ed è noto come la **regola di Hebb**.

**Regola di Hebb:** Se esiste una connessione  $w_{ij}$  tra il neurone  $j$  e il neurone  $i$  e vengono inviati segnali ripetuti dal neurone  $j$  al neurone  $i$ , il che si traduce in entrambi i neuroni attivi contemporaneamente, il peso  $w_{ij}$  viene **rinforzato**. Una possibile formula per la variazione di peso  $\Delta w_{ij}$  è:

$$\Delta w_{ij} = \eta \cdot x_i \cdot x_j$$

con la costante  $\eta$  ((**learning rate**, **tasso di apprendimento**) che determina la dimensione delle singole fasi di apprendimento. Ci sono molte varianti di questa regola, che si traducono in diversi tipi di reti o algoritmi di apprendimento.

## 10.2 Reti di Hopfield

Guardando la regola di Hebb, vediamo che per i neuroni continui  $[0, 1]$  i pesi possono solo crescere con il tempo. Non è possibile che un neurone si indebolisca o addirittura muoia secondo questa regola.

*Regola di Hebb → peso  $w_{ij}$       funzione di attivazione → output  $x_i$*

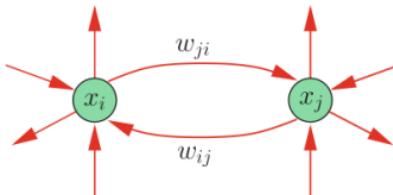
Il problema viene risolto dal modello presentato da Hopfield nel 1982. Utilizza neuroni binari, ma con i due valori -1 per **inattivo** e 1 per **attivo**. Utilizzando la regola di Hebb si ottiene un contributo positivo al peso ogni volta che due neuroni sono attivi contemporaneamente. Se, tuttavia, solo uno dei due neuroni è attivo,  $\Delta w_{ij}$  è negativo.

Le reti di Hopfield, che sono un esempio di **memoria auto-associativa**, si basano su questa idea:

- I modelli possono essere archiviati nella memoria auto-associativa.
- Per richiamare un pattern salvato, è sufficiente fornire un pattern simile. Il magazzino di pattern quindi trova il modello salvato più simile
- Un'applicazione classica di questo modo di operare è il riconoscimento della grafia.

Nella **fase di apprendimento** di una rete Hopfield,  $N$  pattern codificati binari vengono appresi e salvati nei vettori  $\mathbf{q}^1, \dots, \mathbf{q}^N$ . Ciascun componente  $q_i^j \in \{-1, 1\}$  del vettore  $\mathbf{q}^j$  rappresenta un pixel di un pattern. Per i vettori costituiti da  $n$  pixel, viene utilizzata una rete neurale con  $n$  neuroni, uno per ogni pixel. I neuroni sono completamente connessi con la restrizione che la **matrice dei pesi sia simmetrica** e che tutti gli elementi diagonali  $w_{jj}$  siano **zero** → cioè, non c'è connessione tra un neurone e se stesso.

La rete completamente connessa include complessi cicli di feedback, le cosiddette **ricorrenze**, come mostrato nella seguente figura:



È possibile apprendere  $N$  pattern calcolando semplicemente tutti i pesi con la formula:

$$w_{ij} = w_{ji} = \frac{1}{N} \cdot \sum_{k=1}^N q_i^k \cdot q_j^k$$

**NB:** la fase di apprendimento produce una matrice di pesi, ossia tutta l'informazione risiede nei pesi.

**Questa formula mostra una relazione interessante con la regola di Hebb.** Ogni pattern in cui i pixel  $i$  e  $j$  hanno lo stesso valore contribuisce positivamente al peso  $w_{ij}$ , mentre ogni altro pattern dà un contributo negativo. Poiché ogni pixel corrisponde a un neurone, qui vengono rinforzati i pesi tra neuroni che hanno lo stesso valore contemporaneamente.

Una volta che tutti i modelli sono stati memorizzati, la rete può essere utilizzata per il riconoscimento dei modelli. Diamo alla rete un nuovo pattern  $\mathbf{x}$  e aggiorniamo le attivazioni di tutti i neuroni in un processo asincrono secondo la **regola** fino a quando la rete **diventa stabile**, cioè fino a quando le attivazioni non cambiano più.

**Regola:**

$$x_i = \begin{cases} -1 & \text{if } \sum_{j=1, j \neq i}^n w_{ij} \cdot x_j < 0 \\ 1 & \text{else} \end{cases}$$

**HOPFIELDASSOCIATOR( $\mathbf{q}$ )**

**Initialize all neurons:  $\mathbf{x} = \mathbf{q}$**

**Repeat**

**$i = \text{Random}(1, n)$**

**Update neuron  $i$  according to**

**Until  $\mathbf{x}$  converges**

**Return ( $\mathbf{x}$ )**

### 10.3 Memorie associative neurali: esempio riconoscimento facciale

- Una memoria associativa è in grado solo di assegnare il nome giusto alla foto, ma anche a qualsiasi insieme potenzialmente infinito di foto **simili**.
- Una funzione per trovare la somiglianza dovrebbe essere generata da un insieme finito di dati di addestramento, vale a dire le foto salvate etichettate con i nomi. Un approccio semplice per questo è il metodo del **vicino prossimo** introdotto in precedenza. Durante l'apprendimento, tutte le foto vengono semplicemente salvate.
- Per un database con molte foto ad alta risoluzione, questo processo, a seconda della metrica della distanza utilizzata, può richiedere tempi di calcolo molto lunghi e quindi non può essere implementato in questa semplice forma. Pertanto, invece di un algoritmo così pigro, preferiremo uno che trasferisca i dati in una funzione che quindi crea un'associazione molto veloce quando viene applicata.
- Vorremmo che una persona fosse riconosciuta anche se il suo volto appare in un altro punto della foto (traslazione), o se è più piccola, più grande o addirittura ruotata. Anche l'angolo di visione e l'illuminazione potrebbero variare ⇒ **reti neurali**
- Il **modello Hopfield** presentato nel capitolo precedente sarebbe troppo difficile da usare per due ragioni. Primo, è solo una **memoria auto-associativa**, cioè una mappatura approssimativamente identica che mappa oggetti simili all'originale appreso. In secondo luogo, le complesse dinamiche ricorrenti sono spesso difficili da gestire nella pratica.
- **modello di Teuvo Kohonen**, uno dei pionieri in questo settore

### 10.4 Memoria con matrice di correlazione

Kohonen ha introdotto un modello di memoria associativa basato sull'algebra lineare elementare. Questo **mappa** i vettori di input  $\mathbf{q} \in \mathbb{R}^n$  ai vettori di output  $\mathbf{t} \in \mathbb{R}^m$ . Stiamo cercando una matrice  $\mathbf{W}$  che mappi correttamente tutti i vettori di query alle rispettive risposte, basandosi su un insieme di dati di addestramento con  $N$  coppie di **input** (query,  $q$ ), **output** (target,  $t$ ) :

$$T = \{(\mathbf{q}^1, \mathbf{t}^1), \dots, (\mathbf{q}^N, \mathbf{t}^N)\}$$

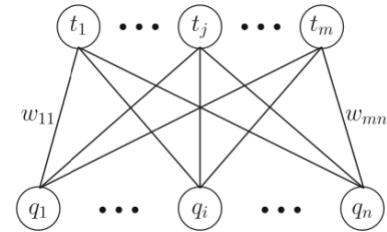
Cioé, per  $p = 1, \dots, N$  deve essere:

$$\mathbf{t}^p = \mathbf{W} \cdot \mathbf{q}^p \quad \text{o, scritto in maniera diversa:}$$

$$\text{Funzione di attivazione: } t_i^p = \sum_{j=1}^n w_{ij} \cdot q_j^p$$

Rappresentazione della memoria associativa Kohonen  
come una rete neurale a due livelli (layers):

$\mathbf{q} \in \mathbb{R}^n$	<i>query</i>
$\mathbf{t} \in \mathbb{R}^m$	<i>target/output desiderabile</i>
$\mathbf{T}$	<i>dataset di training</i>
$\mathbf{W}$	<i>matrice pesi (Obiettivo)</i>
$\mathbf{y}$	<i>output effettivo</i>



Per calcolare i valori  $w_{ij}$  della matrice  $\mathbf{W}$ , usiamo la regola:

$$w_{ij} = \sum_{p=1}^N q_j^p \cdot t_i^p$$

Queste due equazioni lineari possono essere semplicemente intese come una rete neurale a due livelli con  $\mathbf{q}$  come livello di input e  $\mathbf{t}$  come livello di output. I neuroni del livello di output hanno una **funzione di attivazione lineare** e la regola di apprendimento utilizzata, quella per stabilire i pesi delle connessioni, corrisponde esattamente alla regola di Hebb.

**Teorema:** Se tutti gli  $N$  vettori di query  $\mathbf{q}^p$  nei dati di addestramento sono **ortonormali**, ogni vettore  $\mathbf{q}^p$  viene mappato al vettore di destinazione  $\mathbf{t}^p$  tramite moltiplicazione con la matrice  $\mathbf{W}$  definita con il calcolo dei valori  $w_{ij}$  mostrato sopra.

**Definizione:** I vettori  $\mathbf{q}^p$  sono **ortonormali** quando per ogni  $i$  e per ogni  $j \neq i$  si ha che:

$$\mathbf{q}^i \cdot \mathbf{q}^j = 0 \text{ e } |\mathbf{q}^i| = |\mathbf{q}^j| = 1$$

Pertanto, se i vettori di query sono ortonormali, tutti gli input verranno mappati correttamente sui rispettivi output. Tuttavia, l'ortonormalità è una restrizione molto forte. **Poiché le mappature lineari sono continue e iniettive**, sappiamo che la mappatura dai vettori di query ai vettori di destinazione preserva la somiglianza. Le query simili vengono quindi mappate a obiettivi simili a causa della continuità. Allo stesso tempo sappiamo, tuttavia, che query diverse vengono mappate a target diversi.

## 10.5 Reti lineari con errori minimi

La regola Hebb utilizzata nei modelli neurali presentati finora funziona con associazioni tra neuroni vicini. Nella memoria associativa, questo viene sfruttato per apprendere una mappatura dai vettori di query ai target. Questo funziona molto bene in molti casi, specialmente quando i vettori di query sono **linearmente indipendenti**. Se questa condizione non è soddisfatta, ad esempio quando sono disponibili troppi dati di allenamento, sorge la domanda: **come troviamo la matrice del peso ottimale?** Ottimale significa che riduce al minimo l'errore medio.

**Noi umani siamo in grado di imparare dagli errori. La regola di Hebb non offre questa possibilità.** L'algoritmo di **backpropagation**, descritto in seguito, utilizza un'elegante soluzione ispirata all'approssimazione di funzioni per modificare i pesi in modo tale da ridurre al minimo l'errore sui dati di addestramento.

Sia dato un insieme di  $N$  coppie di vettori di allenamento:

$$T = \{(\mathbf{q}^1, \mathbf{t}^1), \dots, (\mathbf{q}^N, \mathbf{t}^N)\}$$

con  $\mathbf{q}^p \in [0, 1]^n$  e  $\mathbf{t}^p \in [0, 1]^m$ . Stiamo cercando una funzione  $f : [0, 1]^n \rightarrow [0, 1]^m$  che minimizza l'errore quadratico sui dati:

$$\sum_{p=1}^N (f(\mathbf{q}^p) - \mathbf{t}^p)^2$$

Supponiamo innanzitutto che i dati **non contengano contraddizioni**, ossia non è presente alcun vettore di query nei dati di addestramento che possa essere mappato su due obiettivi diversi. Definiamo la funzione che azzera anche l'errore sui dati di addestramento (ma non vogliamo un overfitting dei dati tramite memorizzazione):

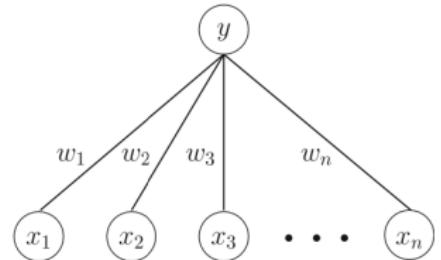
$$\begin{aligned} f(\mathbf{q}) &= 0 && \text{if } \mathbf{q} \notin \{\mathbf{q}^1, \dots, \mathbf{q}^N\} \\ f(\mathbf{q}^p) &= \mathbf{t}^p && \forall p \in \{1, \dots, N\} \end{aligned}$$

## 10.6 Metodo dei minimi quadrati

Data una rete a due strati, il singolo neurone  $y$  del secondo strato calcola la sua attivazione utilizzando:

$$y = f\left(\sum_{i=1}^n w_i \cdot x_i\right) \in \mathbb{R}$$

dove  $f(x) = x$  ossia la funzione identità.



I pesi delle sottoreti sono tutti indipendenti. L'utilizzo di una funzione sigmoide al posto dell'attivazione lineare non offre alcun vantaggio in questo caso perché la funzione sigmoide è strettamente monotonicamente crescente e non cambia la relazione d'ordine tra i vari valori di uscita

**Obiettivo:** cerchiamo un vettore  $\mathbf{w}$  di pesi che minimizza l'errore quadratico:

$$E(\mathbf{w}) = \sum_{p=1}^N (\mathbf{w} \mathbf{q}^p - t^p)^2 = \sum_{p=1}^N \left( \sum_{i=1}^n w_i q_i^p - t^p \right)^2$$

**Minimo:** tutte le derivate parziali devono essere 0, ossia per  $j = 1, \dots, n$ :

$$\begin{aligned} \frac{\partial E}{\partial w_j} &= 2 \cdot \sum_{p=1}^N \left( \sum_{i=1}^n w_i q_i^p - t^p \right) \cdot q_j^p = 0 \\ &\sum_{p=1}^N \left( \sum_{i=1}^n w_i q_i^p \cdot q_j^p - t^p \cdot q_j^p \right) = 0 \\ &\sum_{i=1}^n w_i \sum_{p=1}^N q_i^p \cdot q_j^p = \sum_{p=1}^N t^p \cdot q_j^p \\ A_{ij} &= \sum_{p=1}^N q_i^p \cdot q_j^p \quad \text{and} \quad b_j = \sum_{p=1}^N t^p \cdot q_j^p \end{aligned}$$

*In forma matriciale:*

$$\mathbf{A}\mathbf{w} = \mathbf{b}$$

Queste cosiddette equazioni normali hanno sempre almeno una soluzione quando  $A$  è invertibile. Inoltre, la matrice  $A$  è **definita positiva**, il che implica che la soluzione scoperta nel caso **unico è un minimo globale**.

**Definizione:** Una matrice reale simmetrica  $A$  è  $n \times n$  è **definita positiva** quando, per ogni vettore colonna  $\mathbf{x} \in \mathbb{R}^n$  con  $\mathbf{x} \neq \mathbf{0}$ , si ha  $\mathbf{x}^T \mathbf{A} \mathbf{x} > 0$

## 10.7 La regola del Delta

I minimi quadrati, come il percettrone e l'apprendimento dell'albero decisionale sono degli **algoritmi di apprendimento in batch**, al contrario dell'apprendimento incrementale.

**Nell'apprendimento batch** tutti i dati di addestramento devono essere appresi in un'unica esecuzione. Se vengono aggiunti nuovi dati di allenamento, non possono essere semplicemente appresi in aggiunta a ciò che è già presente. L'intero processo di apprendimento deve essere ripetuto con il set ingrandito.

Questo problema viene risolto da **algoritmi di apprendimento incrementale**, che possono adattare il modello appreso a ogni nuovo esempio aggiuntivo. In quanto segue, **aggioreremo in modo additivo** i pesi per ogni nuovo esempio di allenamento secondo la regola:

$$w_j = w_j + \Delta w_j$$

Deriviamo una variante incrementale del metodo dei minimi quadrati:

$$\frac{\partial E}{\partial w_j} = 2 \cdot \sum_{p=1}^N \left( \sum_{i=1}^n w_i q_i^p - t^p \right) \cdot q_j^p \quad \nabla E = \left( \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right)$$

Il gradiente, sotto forma di vettore di tutte le derivate parziali della funzione di errore, **punta nella direzione del più forte aumento della funzione di errore nello spazio n-dimensionale dei pesi**. Nella **ricerca del minimo**, seguiremo quindi la direzione del gradiente negativo. Come formula per modificare i pesi otteniamo allora:

$$\Delta w_j = -\frac{\eta}{2} \cdot \frac{\partial E}{\partial w_j} = -\eta \cdot \sum_{p=1}^N \left( \sum_{i=1}^n w_i q_i^p - t^p \right) \cdot q_j^p$$

dove il tasso di apprendimento  $\eta$  è una costante positiva liberamente selezionabile. Un  $\eta$  maggiore accelera la convergenza ma allo stesso tempo aumenta il rischio di oscillazione attorno ai minimi o alle valli piane. Considerando che:

$$y^p = \sum_{i=1}^n w_i \cdot q_i^p$$

cioè il p-esimo output  $y^p$  è calcolato moltiplicando per i pesi  $w_i$  l'input di addestramento  $q^p$ , la formula è semplificata e otteniamo la **regola del delta**:

$$\Delta w_j = \eta \cdot \sum_{p=1}^N (t^p - y^p) \cdot q_j^p$$

Ossia, per ogni esempio di addestramento, la differenza tra l'obiettivo  $t^p$  e l'output effettivo  $y^p$  della rete viene calcolata per il dato input  $q^p$ . Dopo aver sommato i risultati di tutti gli  $N$  pattern, i pesi vengono quindi modificati proporzionalmente a tale somma.

```
DELTALEARNING(TrainingExamples,  $\eta$ )
Initialize all weights  $w_j$  randomly
Repeat
     $\Delta w = 0$ 
    For all  $(q^p, t^p) \in TrainingExamples$ 
        Calculate network output  $y^p = \mathbf{w}^p \mathbf{q}^p$ 
         $\Delta w = \Delta w + \eta(t^p - y^p)\mathbf{q}^p$ 
     $w = w + \Delta w$ 
Until  $w$  converges
```

```
DELTALEARNINGINCREMENTAL(TrainingExamples,  $\eta$ )
Initialize all weights  $w_j$  randomly
Repeat
    For all  $(q^p, t^p) \in TrainingExamples$ 
        Calculate network output  $y^p = \mathbf{w}^p \mathbf{q}^p$ 
         $w = w + \eta(t^p - y^p)\mathbf{q}^p$ 
    Until  $w$  converges
```

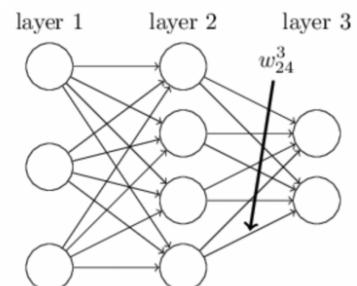
Notiamo che **l'algoritmo non è ancora realmente incrementale** perché i cambiamenti di peso si verificano solo dopo che tutti gli esempi di allenamento sono stati applicati una volta. Possiamo correggere questa carenza modificando direttamente i pesi (discesa gradiente incrementale) dopo ogni esempio di allenamento, il che, in senso stretto, **non è più una corretta implementazione della regola del delta**.

## 10.8 Reti neurali e backpropagation

L'algoritmo ha origine direttamente dalla regola del delta incrementale. Contrariamente alla delta rule, applica una funzione sigmoide non lineare sulla somma ponderata degli ingressi come funzione di attivazione.

**Notazione peso:**  $w_{jk}^l$

connessione dal  $k$ -esimo neurone nel  $(l-1)$  esimo livello  
al  $j$ -esimo neurone nel livello  $l$ -esimo



Nel modello generale di un neurone artificiale, oltre agli stimoli in input provenienti dai neuroni in ingresso, **si aggiunge anche un bias**, ossia una costante additiva caratteristica del neurone stesso. Se specifichiamo il livello  $l$  a cui si trova il neurone, diremo che il neurone  $j$  del livello  $l$  ha un bias  $b_j^l$ .

L'attivazione del neurone  $j$  al livello  $l$ , ossia il suo **output**  $a_j^l$ , è dato dalla funzione di attivazione  $\sigma$  (funzione sigmoidale) applicata alla sommatoria degli stimoli  $a_k^{l-1}$  provenienti dal livello precedente  $(l-1)$ , ciascuno pesato secondo il peso  $w_{jk}^l$  a cui si aggiunge un bias  $b_j^l$

$$a_j^l = \sigma \left( \sum_k w_{jk}^l \cdot a_k^{l-1} + b_j^l \right)$$

<i>matrice di peso per ogni libello <math>l</math></i>	$\mathbf{w}^l$
<i>j-esima riga k-esima colonna</i>	$w_{jk}^l$
<i>vettore di bias per libello <math>l</math></i>	$\mathbf{b}^l$
<i>componente ogni neurone l-esimo livello</i>	$b_j^l$
<i>vettore di attivazione</i>	$\mathbf{a}^l$
<i>componenti di attivazione</i>	$a_j^l$

Infine vettoriazziamo la funzione di attivazione  $\sigma$ , ossia l'applicazione della funzione ad ogni elemento di un vettore  $\mathbf{v} \rightarrow \sigma(\mathbf{v})$ . Le componenti del vettore  $\sigma(\mathbf{v})$  sono  $\sigma(v)_j = \sigma(v_j)$ .

**Forma matriciale:**

$$\begin{aligned}\mathbf{a}^l &= \sigma(\mathbf{w}^l \cdot \mathbf{a}^{l-1} + \mathbf{b}^l) \\ \mathbf{a}^l &= \sigma(\mathbf{z}^l)\end{aligned}$$

**Quantità intermedia: input pesato**

$$\begin{aligned}\mathbf{z}^l &= \mathbf{w}^l \cdot \mathbf{a}^{l-1} + \mathbf{b}^l \\ z_j^l &= \sum_k w_{jk}^l \cdot a_k^{l-1} + b_j^l\end{aligned}$$

## La funzione costo

**L'obiettivo:** della backpropagation è calcolare le derivate parziali di una **funzione di costo**  $C$  rispetto a qualsiasi peso  $w$  o bias  $b$  nella rete:

$$\text{Obiettivo: } \frac{\partial C}{\partial w}, \frac{\partial C}{\partial b}$$

Come esempio di funzione di costo useremo la funzione di costo quadratico:

$$C = \frac{1}{2n} \cdot \sum_x \|y(x) - a^l(x)\|^2$$

<i>numero totale di esempi di training</i>	$n$
<i>esempi di training</i>	$x$
<i>output desiderato corrispondente</i>	$y = y(x)$
<i>numero di livelli nella rete</i>	$L$
<i>vettore delle attivazioni in uscita dalla rete quando <math>x</math> è l'input</i>	$a^L = a^L(x)$

Facciamo due **ipotesi sulla funzione di costo**:

- la funzione di costo possa essere scritta come una media rispetto alle funzioni di costo  $C_x$  per esempi di addestramento individuali  $x$ .

$$C = \frac{1}{n} \sum_x C_x$$

Dato l'esempio avremo che:

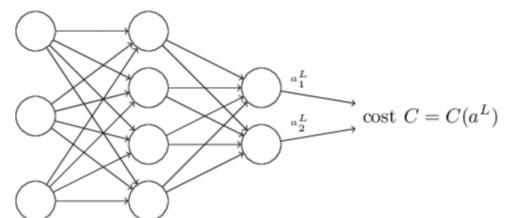
$$C_x = \frac{1}{2}(y - a^l)^2$$

Facendo in questo modo la backpropagation ci consente effettivamente di fare è calcolare le derivate parziali per un singolo esempio di addestramento  $\frac{\partial C_x}{\partial w}, \frac{\partial C_x}{\partial b}$ . Quindi recuperiamo  $\frac{\partial C}{\partial w}, \frac{\partial C}{\partial b}$  calcolando la media sugli esempi di allenamento.

- Il costo può essere scritto in funzione degli output della rete neurale

Ad esempio, la funzione di costo quadratico soddisfa questo requisito, poiché il costo quadratico per un singolo esempio di training  $x$  può essere scritto come

$$C = \frac{1}{2} \|\mathbf{y} - \mathbf{a}^l\|^2 = \frac{1}{2} \sum_j (y_j - a_j^l)^2$$



## Il prodotto di Hadamard o prodotto di Schur

Supponiamo  $s$  e  $t$  siano due vettori della stessa dimensione. Quindi usiamo  $s \odot t$  per denotare il prodotto elemento per elemento dei due vettori.

$$(s \odot t)_j = s_j \cdot t_j$$

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \cdot 3 \\ 2 \cdot 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

## 10.9 Le 4 equazioni fondamentali della backpropagation

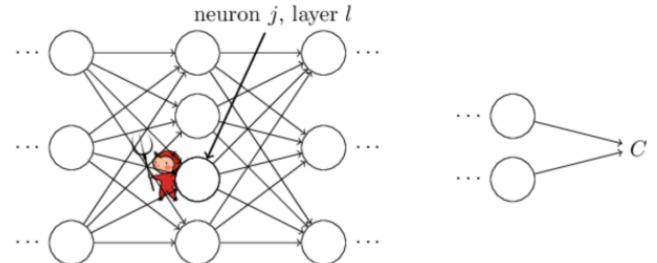
La backpropagation riguarda la comprensione di come la modifica dei pesi e dei bias in una rete modifichi la funzione di costo, questo significa calcolare le derivate parziali

$$\frac{\partial C}{\partial w_{jk}^l}, \frac{\partial C}{\partial b_j^l}$$

Per il calcolo delle derivate parziali viene introdotta una quantità intermedia  $\delta_j^l$ , ossia **errore nel  $j$ -esimo neurone dell' $l$ -esimo livello**. Esempio:

Il demone siede al  $j$ -esimo neurone nel livello  $l$ . Quando arriva l'input per il neurone, il demone interferisce con l'attività del neurone, **aggiungendo un piccolo cambiamento**  $\Delta z_j^l$  all'input pesato del neurone, in modo che invece di emettere

$$\sigma(z_j^l) \xrightarrow{\text{emettiamo}} \sigma(z_j^l + \Delta z_j^l)$$



Questo cambiamento si propaga attraverso gli strati successivi della rete, provocando infine una variazione del costo complessivo di una quantità:

$$\frac{\partial C}{\partial z_j^l} \cdot \Delta z_j^l$$

Supponendo che il **demone sia buono**, ossia sta cercando di **aiutarci a migliorare il costo**, cioè sta cercando  $\Delta z_j^l$  che riduca il costo. Quindi c'è un senso euristico in cui  $\frac{\partial C}{\partial z_j^l}$  è una misura dell'errore nel neurone. Definiamo l'errore  $\delta_j^l$  del neurone  $j$  nel livello  $l$  come:

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$$

La backpropagation si basa su quattro equazioni fondamentali che ci consentono di calcolare  $\delta^l$  e sia il gradiente della funzione di costo. Le quattro equazioni sono indicate di seguito:

- L'equazione per l'errore nello strato di output,  $\delta^L$  - **BP1**
- L'equazione per l'errore  $\delta^l$  in termini di errore nel livello successivo,  $\delta^{l+1}$  - **BP2**
- L'equazione per il tasso di variazione del costo rispetto a qualsiasi bias nella rete - **BP3**
- L'equazione per il tasso di variazione del costo rispetto a qualsiasi peso nella rete - **BP4**

## L'equazione per l'errore nello strato di output, $\delta^L$ - BP1

**Forma non chiusa:**

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \cdot \sigma'(z_j^L)$$

**Forma chiusa:**

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

Osservazioni:

- Il primo termine a dx  $\frac{\partial C}{\partial a_j^L}$  misura quanto velocemente il costo sta cambiando in funzione della  $j$ -esima attivazione dell'output. Se, ad esempio, C non dipende molto da un particolare neurone di output  $j$  allora  $\delta_j^L$  sarà piccolo.
- Il secondo termine a dx  $\sigma'(z_j^L)$  misura la velocità con cui la funzione di attivazione  $\sigma$  cambia in  $z_j^L$ .
- $\nabla_a C$  è definito come un gradiente, ossia un vettore le cui componenti sono le derivate parziali  $\frac{\partial C}{\partial a_j^L}$ . Si può pensare a  $\nabla_a C$  come espressione della velocità di variazione di  $C$  rispetto alle attivazioni dell'output.

## L'equazione per l'errore $\delta^l$ in termini di errore nel livello successivo, $\delta^{l+1}$ - BP2

$$\delta^l = \left( (w^{l+1})^T \cdot \delta^{l+1} \right) \odot \sigma'(z^l)$$

Osservazioni:

- Supponiamo di conoscere l'errore  $\delta^{l+1}$  al livello  $(l + 1)$ . Quando applichiamo la matrice trasposta del peso possiamo pensare intuitivamente a questo come a spostare l'errore all'indietro attraverso la rete, dandoci una sorta di misura dell'errore all'uscita dell' $l$ -esimo livello.
- $\odot \sigma'(z^l)$  sposta l'errore indietro attraverso la funzione di attivazione nel livello  $l$ , dandoci l'errore  $\delta^l$  in ingresso al livello  $l$ .
- Combinando (BP2) con (BP1) possiamo calcolare l'errore  $\delta^l$  per ogni livello della rete. Iniziamo usando (BP1) per calcolare  $\delta^L$  e quindi applichiamo l'equazione (BP2) per calcolare  $\delta^{L-1}$  e così via, per tutto il percorso attraverso la rete.

## L'equazione per il tasso di variazione del costo rispetto a qualsiasi bias nella rete - BP3

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad \rightarrow \quad \frac{\partial C}{\partial b} = \delta$$

Osservazioni

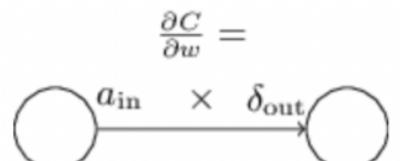
- L'errore  $\delta_j^l$  è esattamente uguale alla velocità di variazione  $\frac{\partial C}{\partial b_j^l}$

## L'equazione per il tasso di variazione del costo rispetto a qualsiasi peso nella rete - BP4

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \cdot \delta_j^l \quad \rightarrow \quad \frac{\partial C}{\partial w} = a_{in} \cdot \delta_{out}$$

Osservazioni

- Dove si intende che  $a_{in}$  è l'attivazione del neurone in input alla connessione con peso  $w$  e  $\delta_{out}$  è l'errore del neurone in output alla connessione con peso  $w$ .



- Una conseguenza di questa equazione è che quando l'attivazione  $a_{in}$  è piccola ( $a_{in} \approx 0$ ) anche il termine gradiente  $\frac{\partial C}{\partial w}$  tenderà ad essere piccolo. In questo caso, diremo che il peso impara lentamente, il che significa che non cambia molto durante la discesa del gradiente. In altre parole, una conseguenza di (BP4) è che i pesi che si trovano a valle di neuroni a bassa attivazione apprendono lentamente.

## Info generali

- Considerando il termine  $\sigma'(z_j^L)$  in **(BP1)** e sapendo che la  $\sigma$  sigmoide diventa piatta agli estremi quindi alta attivazione ( $\approx 1$ ) o bassa attivazione ( $\approx 0$ )  $\Rightarrow \sigma'(z_j^L) \approx 0$ .

$$(\approx 0) \vee (\approx 1) \Rightarrow \sigma'(z_j^L) \approx 0$$

Ossia il neurone di output si è saturato e, di conseguenza, il peso ha smesso di apprendere (o sta imparando lentamente).

- Considerando il termine  $\sigma'(z_j^l)$  in **(BP2)**.  $\delta_j^l$  diventa piccolo se il neurone è vicino alla saturazione. E questo, a sua volta, significa che qualsiasi peso in input a un neurone saturo imparerà lentamente.
- **NB:** Le quattro equazioni fondamentali risultano essere valide per qualsiasi funzione di attivazione, non solo per la funzione sigmoide standard. In questo modo possiamo usare queste equazioni per progettare funzioni di attivazione che hanno particolari proprietà di apprendimento desiderate.

$$\begin{aligned} BP1 \quad & \delta^L = \nabla_a C \odot \sigma'(z^L) \\ BP2 \quad & \delta^l = \left( (w^{l+1})^T \cdot \delta^{l+1} \right) \odot \sigma'(z^l) \\ BP3 \quad & \frac{\partial C}{\partial b_j^l} = \delta_j^l \\ BP4 \quad & \frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \cdot \delta_j^l \end{aligned}$$

**Osservazione:** Tutte e quattro le equazioni sono conseguenze della regola della catena (**chain rule**) del **calcolo multivariabile**, una regola di derivazione che permette di calcolare la derivata della funzione composta di due funzioni derivabili. (concatenazione di derivate)

## 11 Dimostrazioni per esame

- Illustrare con pseudocodice l'algoritmo K-means:

- Il numero di cluster è noto a priori  $k$
- Vengono scelti  $k$  centroidi in maniera casuale o manuale
- i punti  $x$  vengono classificati al cluster più vicino  $C_i$  in base alla distanza dal centroide
- Vengono ricalcolati i centroidi dopo la nuova classificazione sommando vettorialmente i punti

$$\mu = \frac{1}{l} \cdot \sum_{i=1}^l x_i$$

- Questo algoritmo non garantisce la convergenza, ma di solito converge molto rapidamente e il numero di iterazioni necessarie è minore del numero di punti
- La complessità è  $O(n \cdot d \cdot k \cdot t)$  - numero di dati - dimensionalità dello spazio delle caratteristiche - numero cluster definiti - numero iterazioni

```

K-MEANS( $x_1, \dots, x_n, k$ )
initialize cluster centers  $\mu_1 = x_{i_1}, \dots, \mu_k = x_{i_k}$  (e.g. randomly)
Repeat
    classify  $x_1, \dots, x_n$  to each's nearest  $\mu_i$ 
    recalculate  $\mu_1, \dots, \mu_k$ 
Until no change in  $\mu_1, \dots, \mu_k$ 
Return( $\mu_1, \dots, \mu_k$ )

```

- Il paradosso del barbiere: *C'è un barbiere che rade tutti quelli che non si radono.*

*La frase tradotta in formula del prim'ordine diventa:*

<i>Barbiere</i>	<i>costante</i>
<i>Rade</i>	<i>predicato, 1° chi rade, 2° il rasato</i>
<i>Rade(x, x)</i>	<i>rado me stesso</i>

$$KB = \forall x (Rade(\text{Barbiere}, x) \Leftrightarrow \neg Rade(x, x))$$

*trasformata in clausole, dà il seguente insieme di clausole, il quantificatore universale è sottointeso, ossia tutte le variabili sono quantificate:*

$$\forall x \left( (Rade(\text{Barbiere}, x) \Rightarrow \neg Rade(x, x)) \wedge (\neg Rade(x, x) \Rightarrow Rade(\text{Barbiere}, x)) \right)$$

$$\forall x \left( (\neg Rade(\text{Barbiere}, x) \vee \neg Rade(x, x)) \wedge (Rade(x, x) \vee Rade(\text{Barbiere}, x)) \right)$$

$$\begin{array}{ll} \neg Rade(\text{Barbiere}, x) \vee \neg Rade(x, x) & [\text{clausola 1}] \\ Rade(x, x) \vee Rade(\text{Barbiere}, x) & [\text{clausola 2}] \end{array}$$

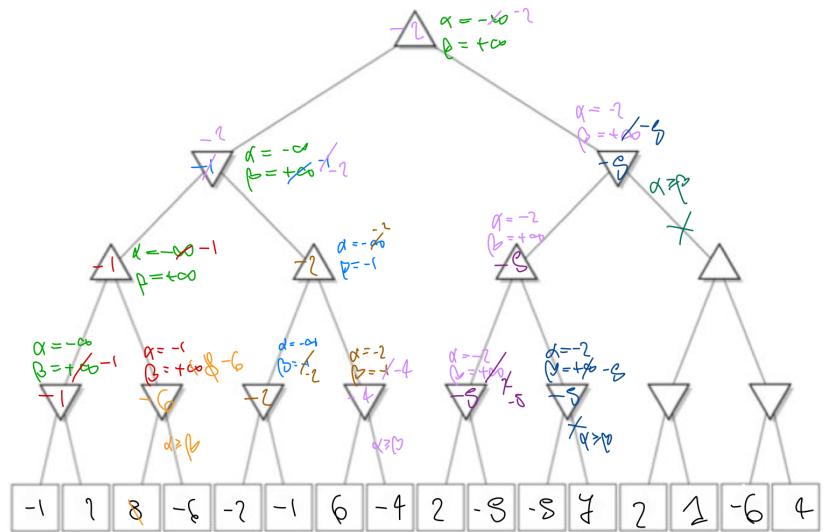
*Applicando la risoluzione viene eliminata una letterale per volta, dunque otteniamo:*

$\neg Rade(\text{Barbiere}, x) \vee \neg Rade(x, x)$	<i>[clausola 1]</i>
$Rade(x, x) \vee Rade(\text{Barbiere}, x)$	<i>[clausola 2]</i>
$\neg Rade(\text{Barbiere}, \text{Barbiere})$	<i>[clausola 3, per fattorizzazione su 1, con <math>\sigma = [x/\text{Barbiere}]</math>]</i>
$Rade(\text{Barbiere}, \text{Barbiere})$	<i>[clausola 4, per fattorizzazione su 2, con <math>\sigma = [x/\text{Barbiere}]</math>]</i>
{}	<i>[risoluzione tra 3 e 4] c.v.d</i>

- Minimax  $\alpha/\beta$ 
  - Partendo dal nodo radice e su tutti i nodi più a sx si scrive  $\alpha = -\infty$  e  $\beta = +\infty$
  - In base a chi tocca Min/Max si itera sulle foglie e si scrive in  $Minimo = \beta$ / $Massimo = \alpha$  ad ogni iterazione. Finite le iterazioni il valore del nodo  $V$  risulta essere l'ultimo valore che ha aggiornato  $\beta/\alpha$
  - Un volta fatti tutti i successori del nodo si sale al nodo padre si aggiorna  $\alpha/\beta$  in base a chi tocca con il valore  $V$  precedente e si riscende a dx con il nuovo  $\alpha/\beta$  modificato precedentemente
  - La condizione di potatura è  $\alpha \geq \beta$

```

function alphabeta(node,  $\alpha$ ,  $\beta$ , MAX) is
  if isLeaf(node) == true then
    return util(node)
  if MAX == true then
     $v = -\infty$ 
    for each node.child do
       $v = \max(v, alphabeta(child, \alpha, \beta, FALSE))$ 
      if  $v \geq \beta$  then
        break
       $\alpha = \max(\alpha, v)$ 
    return v
  else
     $v = +\infty$ 
    for each node.child do
       $v = \min(v, alphabeta(child, \alpha, \beta, TRUE))$ 
      if  $v \leq \alpha$  then
        break
       $\beta = \min(\beta, v)$ 
    return v
  
```



- Definizione ricorsiva specificando caso base, ipotesi ricorsiva e passo della funzione APPEND

```

%KB
append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).

%Query
?- trace, append([a, b, c], [d, 1, 2], Z).

%Risultato:
Call:append([a, b, c], [d, 1, 2], _4306)
Call:append([b, c], [d, 1, 2], _692)
Call:append([c], [d, 1, 2], _698)
Call:append([], [d, 1, 2], _704)
Exit:append([], [d, 1, 2], [d, 1, 2])
Exit:append([c], [d, 1, 2], [c, d, 1, 2])
Exit:append([b, c], [d, 1, 2], [b, c, d, 1, 2])
Exit:append([a, b, c], [d, 1, 2], [a, b, c, d, 1, 2])
Z = [a, b, c, d, 1, 2]

```

```

append([] ,L,L). % Caso base = termina ricorsione
append([X|L1] , L2, [X|L3]) :- ... % Passo della funzione
append([X|L1] , L2, [X|L3]) :- append(L1,L2,L3). % ipotesi ricorsiva

```