
Gianfranco Basti

# ETICA DEL MACHINE LEARNING

## PARTE II:
## ALGORITMI DEL MACHINE LEARNING.
## IA DISCRIMINATIVA E IA GENERATIVA

Roma 2025

# SOMMARIO

# Schema del Corso

**PARTE PRIMA: LE ORIGINI DELL'IA E DEL MACHINE LEARNING**

1. Dalla Logica all'Informatica: il Calcolatore Universale
2. Il Test di Turing e il Programma di Ricerca dell'IA Simbolica
3. La Nascita delle Scienze e Neuroscienze Cognitive
4. L'IA Pre-Simbolica e le Reti Neurali
    a. IA Discriminativa e IA Generativa
    b. Reti Neurali Naturali
    c. Reti Neurali Artificiali
    d. Il Machine Learning

**PARTE SECONDA: ALGORITIMI DI MACHINE LEARNING.**
**IA DISCRIMINATIVA E IA GENERATIVA**

1. Machine Learning Supervisionato, Con Rinforzo, Non-Supervisionato,
2. Algoritmi di Machine Learning nell'IA Discriminativa: il Deep-Learning
    a. Il Perceptrone Multistrato

b. Machine Learning Supervisionato: la Back-Propagation
c. Machine Learning Supervisionato: Reti Convolutive
d. Machine Learning Non-Supervisionato
e. Machine Learning Quantistico

3. IA Generativa: la Rivoluzione dei Transformers nel Machine Learning

**PARTE TERZA: ETICA E MACHINE LEARNING**

1. Etica nell'IA ed Etica nel Machine Learning
2. Le "Ingiustizie Algoritmiche" nel Machine Learning
3. Problemi Etici dell'IA Generativa
4. Neuroetica nelle Neuroscienze Cognitive ed Etica nel Machine Learning
5. Logica Deontica e la Nozione di "Algoritmo Buono" nel Machine Learning

**CONCLUSIONI**

- Responsabilità etica condivisa uomo-macchina nei sistemi di IA autonomi e non-autonomi

# LISTA DEGLI ACRONIMI PRINCIPALI

(Di solito usiamo l'acronimo inglese più diffuso in letteratura con traduzione italiana)

| | |
|---|---|
| BP | *Back-Propagation* (Propagazione All'Indietro dell'Informazione |
| FFN | *Feed-Forward Network* (Reti con Propagazione In Avanti dell'Informazione) |
| LLM | *Large Linguistic Models* (Modelli Linguistici di Grandi Dimensioni) |
| LSTM | *Long-Short-Term Memories* (Reti Neurali a Memoria a Breve e Lungo Termine) |
| ML | *Machine Learning* (Apprendimento Automatico) |
| NLP | *Natural Language Processing* (Processazione di Linguaggi Naturali) |
| RNA | Reti Neurali Artificiali (nelle macchine) |
| RNC | Reti Neurali Convolutive |
| RNN | Reti Neurali Naturali (nel cervello) |
| RNR | Reti Neurali Ricorrenti con Feed-Forward e Feed-Backward dell'Informazione |

# 4. Tre tipi di apprendimento automatico

## 4.1. Dalla scienza dei dati all'apprendimento automatico (ML)

- La scienza dei dati (*data science*) è una disciplina finalizzata a unificare metodi statistici, analisi dei dati, apprendimento automatico e metodi correlati al fine di comprendere e analizzare i fenomeni reali attraverso i dati.
- L'apprendimento automatico (*machine learning*) è lo studio scientifico di algoritmi e modelli statistici che i sistemi informatici utilizzano per eseguire un compito specifico senza utilizzare istruzioni esplicite, basandosi invece su modelli e inferenze. È visto come un sottoinsieme dell'intelligenza artificiale, la cosiddetta "IA non-simbolica".
- **Differenze:**
- Come si può vedere, la scienza dei dati non è una singola disciplina ma una raccolta di vari campi di studio come l'analisi dei dati, l'apprendimento automatico, le statistiche, le visualizzazioni che vengono utilizzate per comprendere i dati e risolvere i problemi utilizzando modelli statistici o di apprendimento automatico.

- D'altra parte, l'apprendimento automatico è un campo a sé stante che utilizza vari algoritmi per creare modelli di ML in grado di eseguire attività senza programmazione esplicita. Infatti, come abbiamo visto sopra, l'apprendimento automatico è una delle competenze utilizzate nella scienza dei dati. Questo è il motivo per cui troviamo così tante somiglianze tra i due.
- Competenze comuni. Sia l'apprendimento automatico che la scienza dei dati comprendono:
- Algoritmi di regressione e classificazione dell'apprendimento supervisionato per la creazione di modelli predittivi.
- Algoritmi di apprendimento non supervisionato per trovare strutture nascoste, associazioni, correlazioni e valori anomali nei dati.

## 4.2.Tre fasi in ogni sistema di ML su banche-dati estese (*big data*)

- Come premessa a tutto quanto diremmo in seguito sugli algoritmi di ML esiste un ulteriore legame fa la **scienza statistica dei dati** e **apprendimento automatico** su **banche-dati particolarmente estese**.

- Per intenderci, con miliardi di dati e decine di miliardi di parametri (*big data*). Dove cioè **il supporto decisionale di sistemi di AI con ML è assolutamente indispensabile,** perché i big-data sono intrattabili da qualsiasi **esperto umano** e conseguentemente per qualsiasi **sistema esperto dell'AI simbolica** senza ML che nel programma automatizzano alberi inferenziali di scelta dell'esperto umano.
- Data questa condizione di intrattabilità sistematica dei big-data dagli umani, ogni algoritmo di ML su big-data si sviluppa **secondo tre fasi:**

1. **Fase di allenamento (*training phase)***. Si prende dall'intera base-dati (*database)* un sottoinsieme di essa di cui già si conosce quale sia la **classificazione corretta** e su di essa si allena l'algoritmo di ML finché esso, adattando automaticamente i suoi parametri sulle correlazioni nascoste presenti nel campione, non sia in grado anch'esso di classificarli allo stesso modo con un errore minimo.
2. **Fase di prova (*testing phase).*** Fissati i parametri dell'algoritmo di ML in base all'insieme di allenamento, si sceglie un ulteriore campione di dati di cui si conosce la classificazione corretta ma che il sistema non ha visionato nel suo training e si controlla se effettivamente l'algoritmo di ML è in grado di riconoscerli con un errore accettabile.

3. **Fase di applicazione (*application phase).*** Si applica l'algoritmo di ML allenato e provato sui rispettivi campioni sul resto del database **"sconosciuto" all'algoritmo** confidando nella sua capacità di classificazione in base al training/testing precedente.

- ♦ Come ci saremo accorti **fondamentale per il ML** è supporre che i campioni della fase di training e della fase di testing dell'algoritmo di ML siano **rappresentativi dell'intera base di dati** su cui l'algoritmo di ML sarà applicato.
- ♦ Naturalmente esistono diversi metodi statistici per garantire **la rappresentatività di un campione** rispetto all'intera base di dati.
- ♦ In ogni caso, dato tutto questo, **due sono gli errori opposti** in cui gli algoritmi di ML sono esposti durante la fase di training. Essi vanno sotto il nome di modelli **super-corrispondenti (*overfitting*)** e **sub-corrispondenti (*underfitting*)** al training set.

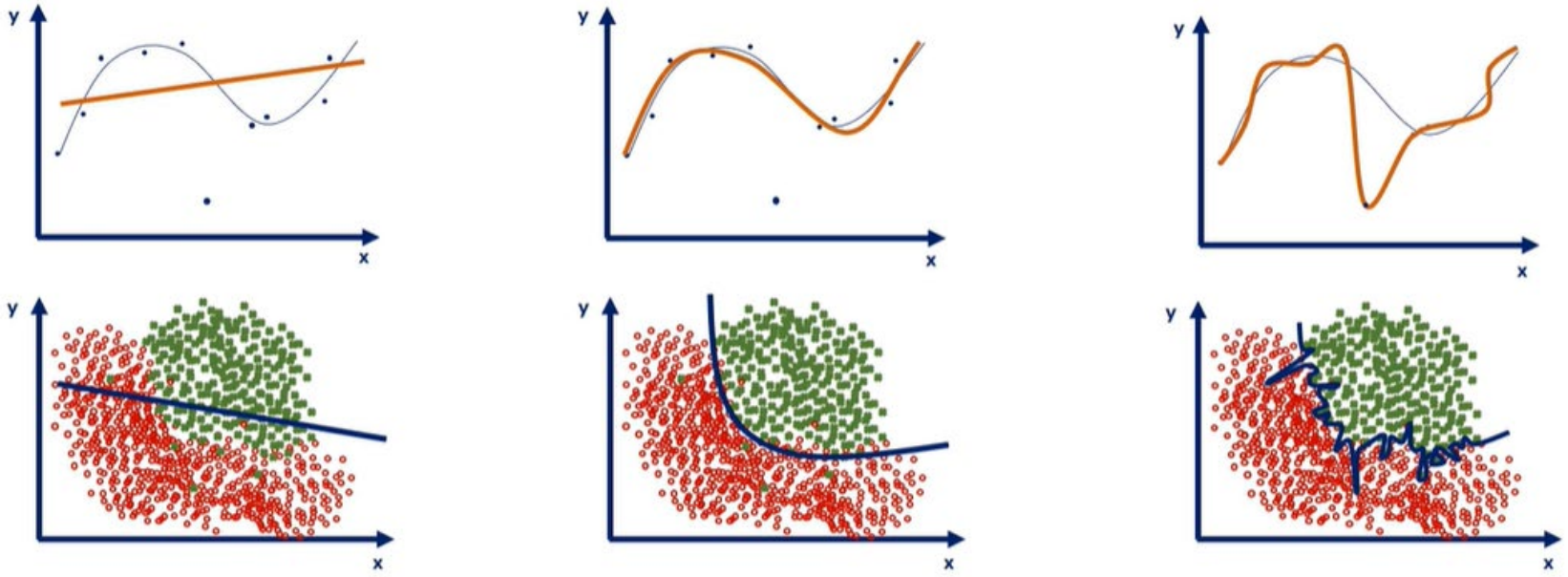



*Figura 1. Esempi di modelli:* ***Sinistra****: underfitted;* ***Destra****: overfitted;* ***Centro****: buoni, applicati sotto ad un compito di classificazione fra 2 classi di oggetti (verdi=cani e rossi=gatti), un classico esempio di funzione XOR.*

- I due esempi a sinistra di **underfitting** evidenziano bene che lo **XOR non è una funzione linearmente separabile,** ovvero la separatrice fra le due classi non può essere una retta. Infatti, se guardiamo all'esempio di classificazione underfitted in basso vediamo che un numero alto di gatti (nella parte alta della separatrice) sono

finiti fra i cani e un numero alto di cani (nella parte centrale della separatrice) sono finiti fra i gatti.

- D'altra parte, i due esempi a destra di **overfitting** evidenziano un modello di classificazione troppo dipendente dal training set. La separatrice, infatti, è talmente precisa da seguire anche i **dati casualmente distribuiti** inevitabilmente presenti in qualsiasi campione statistico (i pochi cani nella parte bassa della distribuzione e i pochi gatti nella parte alta). Questo significa che quel modello non sarà in grado di discriminare adeguatamente su dati diversi da quelli del campione su cui è stato fatto il training.
- Il modello corretto con buone capacità di generalizzazione al resto del database (nella supposizione che il training set sia un campione rappresentativo del dataset) è dunque quello nelle due figure al centro, dove nel caso in basso dello **XOR fra due classi** si evidenzia il fatto che esso è qui una **funzione non-lineare quadratica** (una parabola).

## 4.3. ML supervisionato, non-supervisionato, con rinforzo

- ML supervisionato. L'apprendimento supervisionato è una branca dell'apprendimento automatico in cui il modello **si allena (*training*)** sui dati storici per apprendere la relazione tra dati di input e output e creare una funzione di mappatura. Il modello utilizza quindi questa funzione di mappatura per prevedere o classificare l'output di nuovi dati di input non ancora visionati.
- ML non-supervisionato. L'apprendimento non-supervisionato è una branca dell'apprendimento automatico in cui il sistema impara a riconoscere un modello o struttura di correlazioni nascoste all'interno di dati che non sono stati ancora classificati. Non prevede, cioè, alcuna fase di allenamento su basi di dati precedentemente classificati come nell'apprendimento supervisionato.
- ML con rinforzo. È una branca dell'apprendimento automatico in cui la macchina apprende il suo ambiente interagendo con esso senza il coinvolgimento di alcun essere umano. Nel perseguire **il raggiungimento dell'obiettivo** impara attraverso una serie di azioni, ottenendo ricompense o punizioni in cambio della sua azione.

- **E’ chiaro che questo tipo di ML può diventare un metodo immediato di implementazione di algoritmi deontici di ML: nell’allenamento della macchina sulle serie storiche basta premiare decisioni del sistema** che soddisfano, o viceversa punire decisioni del sistema che non soddisfano, determinati vincoli etici.

# 5. Algoritmi di Machine Learning: il Deep Learning[1]

## 5.1. Il Perceptrone multistrato

### 5.1.1. Reti di McCulloch e Pitts *versus* Perceptrone di Rosenblatt

- Come abbiamo già anticipato nella Prima Parte (§3.4.4) il progresso del perceptrone di Rosenblatt (Rosenblatt, 1961) rispetto alle reti di McCulloch & Pitts (McCulloch & Pitts, 1943) è che esso sia in grado di implementare mediante i suoi **neuroni lineari** un calcolo parallelo per funzioni di **riconoscimento automatico di oggetti** capaci di combinare insieme diverse caratteristiche dell'input.
- Sebbene il fatto di implementare semplici funzioni di attivazione lineari come i neuroni di McCulloch e Pitts fa sì che anche il perceptrone non sia in grado di implementare lo XOR e quindi non sia in grado di **classificare diversi oggetti** come invece un classico computer programmabile è in grado di fare**.**

- È questo il cuore della critica che Marvin Minsky e Seymour Papert fecero nel 1969 al perceptrone di Rosenblatt (Minsky & Papert, 1987) che fu talmente distruttiva verso l'approccio delle RNA che ne bloccò di fatto lo sviluppo per quasi vent'anni finché l'uso di un modello di neurone artificiale con **funzione di attivazione sigmoidale non-lineare** non risolse in linea di principio il problema (cfr., sotto § 5.2)
- Come anticipato sempre nella Prima Parte, il segreto della capacità di riconoscimento del perceptrone è nascosto nel fatto che il supporto su cui le funzioni del perceptrone sono definite è costituito da un **filtro** che è di fatto **un'unione di sottoinsiemi disgiunti dello spazio di input.**

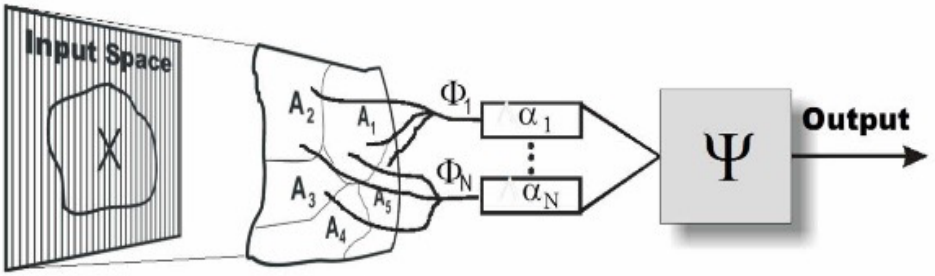



*Figura 2. Rappresentazione schematica dell'architettura parallela del perceptrone lineare, dove ogni neurone di ingresso* $\boldsymbol{\alpha_i}$ *calcola indipendentemente una funzione diversa* $\boldsymbol{\Phi_i}$*, i*

*cui supporti (dominio-codominio) sono definiti su un filtro costituito dall'unione di sottoinsiemi disgiunti $\boldsymbol{A_i}$ dello spazio di input ("retina") su cui è raffigurato un dato pattern di input **X**. Il neurone di output, quindi, calcola la semplice sommatoria $\boldsymbol{\Psi}$ dei risultati dei calcoli dei neuroni di input. Si tratta dunque di un'architettura lineare senza strati intermedi fra i neuroni di input e quello di output.*

♦ I limiti di una rete di neuroni di McCulloch & Pitts sono così sintetizzabili:

1. **Il neurone di McCulloch-Pitts poteva prendere solo valori booleani 1/0 come input.** Ma i problemi del mondo reale non si limitano ai soli valori booleani. Ad esempio, non è possibile fornire a un neurone siffatto come input numeri di variabili reali come età, prezzo, area, volume, ecc.
2. **Non considera i pesi assegnabili agli input**. I pesi, nel senso lineare della **frequenza relativa** (medie) di attivazione sono molto importanti per indicare quale delle caratteristiche di input svolge un ruolo più importante nell'output e quali caratteristiche svolgono un ruolo molto limitato. Quindi senza pesi non può determinare l'importanza, il **peso statistico** dei dati di input, come invece la **legge di Hebb** evidenzia che è la regola nelle RNR (cfr. I Parte, §3.4.2).

3. **L'unico parametro che c'è in questo modello è la soglia Θ**. Ma anche in questo caso non c'è alcun tipo di apprendimento per determinare il valore ottimale della soglia sulla base dei dati passati. Quindi questo tipo di neurone non implementa alcun modello di apprendimento automatico.

♦ Viceversa, nel perceptrone di Rosenblatt:

1. **Gli input al perceptrone non sono più limitati a valori booleani**. I valori di input possono essere rappresentati da qualsiasi numero reale.
2. **Ogni input ha un peso probabilistico (frequenza) associato ad esso nel perceptrone**. Questi pesi sono inizialmente sconosciuti (ovvero assegnati in modo casuale) ma vengono appresi dal perceptrone durante la fase di allenamento.
3. **La funzione di attivazione del perceptrone in output Ψ** è data perciò dalla sommatoria dei suoi input **moltiplicati per i relativi pesi $\Phi_i$** lungo le diverse linee di calcolo corrispondenti ai neuroni di input $\alpha_i$. Si tratta quindi sempre di una **funzione a gradino di Heaviside (cfr.** Figura 10**)** il cui output è sempre un **valore discreto 1/0** (= oggetto riconosciuto/non-riconosciuto) conseguente però ad un vero e proprio apprendimento automatico da parte del perceptrone nel determinare i pesi.

♦ I passi di calcolo del perceptrone dopo l'apprendimento sono perciò i seguenti:

1. I dati di input sono abilitati nel Perceptrone.
2. Applica il prodotto tra gli input e i loro rispettivi pesi e poi sommali.
3. Applica la funzione di soglia a gradino sulla sommatoria precedente:
    a. Se l'output della funzione a gradino $\boldsymbol{\Theta \geq 0}$, allora viene attivato il perceptrone: $\boldsymbol{\Psi = 1}$.
    b. Se l'uscita della funzione a gradino $\boldsymbol{\Theta < 0}$, allora il perceptrone non si attiva: $\boldsymbol{\Psi = 0}$.

♦ Tuttavia, per capire quale sia il vero **valore aggiunto** del ML del perceptrone per il **riconoscimento automatico di oggetti** occorre approfondire la **teoria dei filtri** nell'ambito della teoria degli insiemi, visto che il cuore di questa architettura di calcolo consiste proprio nel fatto che i supporti dei neuroni di input sono definiti su **un filtro dello spazio (insieme) di input.**

### 5.1.2. Il perceptrone lineare multistrato di Gamba

♦ Un valore aggiunto che si amplifica ulteriormente quando da un architettura ad un solo strato quale è l'originario **perceptrone di Rosenblatt** passiamo all'architettura

**multistrato del perceptrone di Gamba** che aggiunge all'originale **strato iniziale** di neuroni di input e **strato finale** del neurone di output dell'architettura di Rosenblatt uno **strato intermedio** di neuroni o **strato nascosto (*hidden layer*)** che possono calcolare diversi **possibili riconoscimenti dell'oggetto** così che il neurone di output può riconoscere in base al calcolo **dell'errore quadratico medio (*mean squared error*)** di ciascun diverso possibile riconoscimento nei neuroni intermedi. Ovvero:

- Se un vettore di previsioni *n* viene generato da un campione di punti dati *n* sulle variabili, *Y* è il vettore dei valori osservati della variabile prevista, $\hat{Y}$ è il vettore dei valori previsti, allora l'MSE all'interno del campione è calcolato come: $1/n \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$. In pratica, l'MSE è la media $1/n \sum_{i=1}^{n}$ dei quadrati degli errori $(Y_i - \hat{Y}_i)$. L'MSE una grandezza media molto facile da calcolare ma che dipende criticamente dal campione prescelto per effettuare la stima.

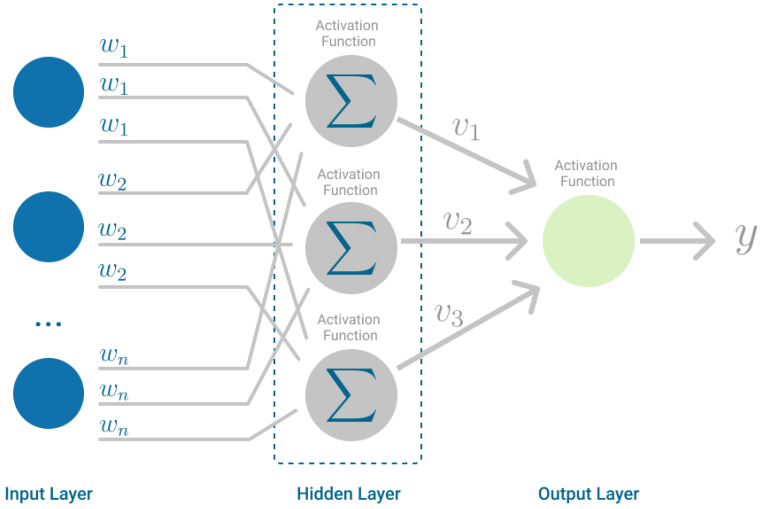



*Figura 3. Schema di un perceptrone multistrato di Gamba.*

### 5.1.3. La teoria insiemistica dei filtri: dal perceptrone alle reti convolutive

♦ Sintetizzando quanto diremo nel resto di questa Seconda Parte, possiamo dire:

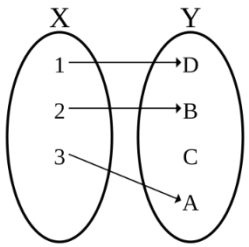
1. Se la presenza di **filtri** nell'algoritmo di ML del perceptrone unistrato o multistrato è il segreto in cui consiste la sua performance nel **riconoscimento 1/0 di oggetti** sulla base di un campione dei medesimi su cui si allena la rete;
2. La presenza di una **funzione di attivazione non-lineare** che renda capace la rete di calcolare lo XOR è il segreto delle **reti multistrato supervisionate** come **classificatori di oggetti**. Reti cioè il cui output 1/0 consista (in una stima della probabilità) dell'appartenenza di un oggetto **all'una o all'altra classe di quelle previste.**

♦ La potenza delle **reti supervisionate convolutive** da cui dipende l'attuale esplosione dell'uso degli algoritmi di ML in IA dipende dal fatto che nel loro algoritmo di ML amplificano di molto la loro **capacità di assegnare gli oggetti alle diverse classi previste** nel loro apprendimento supervisionato, perché, diversamente dall'algoritmo di ML della **back-propagation** che costituisce il progenitore di tutte le reti supervisionate, includono la possibilità di usare **filtri adattivi** nella funzione di riconoscimento, andando dunque al di là delle capacità del perceptrone che usa filtri **predefiniti**.

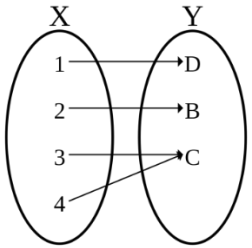
### 5.1.4. Mappe iniettive, suriettive e biiettive

- Per capire cosa sia un filtro inteso come **unione di disgiunti** o **somma disgiunta** in Teoria degli Insiemi (TI), o **colimite** in Topologia e Teoria delle Categorie (TC) o **kernel** in informatica, torniamo ad alcune nozioni elementari di TI.
- È molto importante non confondere l'uso delle medesime nozioni/termini in TI e in TC, anche se l'esemplificazione insiemistica di nozioni di TC può aiutare. Per esempio, **morfismo** in TC è sinonimo di **freccia,** ovvero di relazione orientata da un dominio (*source*) a un codominio (*target*) di oggetti qualsiasi. In TI morfismo è sinonimo di **funzione** ovvero di "relazione che pone in corrispondenza (*mapping*) **insiemi ordinati (**ovvero fra i quali sia definita una **relazione di ordinamento** ($\leq$)) di oggetti" generalmente **insiemi numerici**. I principali morfismi in questo senso (= funzioni, $f: X \rightarrow Y$) sono:

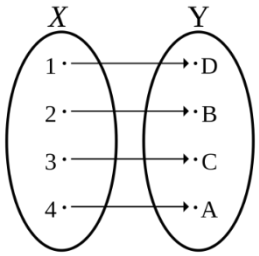
1. **Iniettivi (uno-al-massimo-uno):** ovvero iniettivi senza essere suriettivi, se ogni elemento del codominio è relato al massimo con un elemento del dominio




2. **Suriettivi (uno-al-minimo-uno):** ovvero suriettivi senza essere iniettivi, se ogni elemento del codominio è relato almeno ad un elemento del dominio:




3. **Biiettivi (iniettiva e suriettiva):** se ogni elemento del codominio è relato esattamente a un elemento del dominio (= uno-a-uno senza essere biunivoca):




- Da queste semplicissime definizioni si può comprendere la nozione di **omomorfismo** sia in TC che in TI: un **omomorfismo** è un **morfismo (mappa) che mantiene la struttura.** Ciascuna delle tre collezioni di morfismi (mappe) esaminate finora in TI sono altrettanti **omomorfismi**.
- Generalmente esiste una stretta relazione fra **biiettività e cardinalità degli insiemi:** due insiemi, fra i quali è possibile definire una relazione biiettiva fra i loro elementi, hanno **la medesima cardinalità** e se omomorfi saranno anche **isomorfi.**
- Un caso interessante è la composizione biiettiva $g \circ f$ ($X \to Z$) fra due funzioni, iniettiva $f$: $X \to Y$ e suriettiva: $g$: $Y \to Z$), dove cioè il codominio di una, $f$, è il dominio dell'altra, $g$.

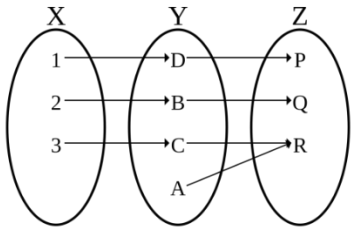



*Figura 4. Relazione biiettiva (XZ) come composizione di una iniettiva (XY) e una suriettiva (YZ).*

♦ La prima relazione XY non necessita di essere suriettiva e la seconda YZ non necessita di essere iniettiva; quindi, gli insiemi considerati **non hanno la medesima cardinalità** (numero di elementi), sebbene il punto di partenza e di arrivo X e Z della relazione composta soddisfino una **relazione biiettiva** e quindi abbiano la medesima cardinalità.

### 5.1.5. Utilizzo del metalinguaggio della Teoria delle Categorie in logica e conseguenze per l'informatica e le scienze cognitive

♦ Nella **categoria degli insiemi, Set,** nel metalinguaggio della **Teoria delle Categorie** (TC) che ha come **primitivo** non **l'appartenenza** ∈ (*membership* "essere membro/elemento di") come la **Teoria degli Insiemi,** ma il **morfismo o "freccia"** → (ogni **categoria** in TC si definisce come **una collezione di morfismi, oggetti e**

**composizioni di morfismi che mantiene la struttura.** Così **Set** è la categoria che ha **insiemi** come oggetti e **funzioni** come morfismi) le relazioni iniettiva, suriettiva, biiettiva, corrispondono a tre tipi diversi di **omomorfismo ("freccia che mantiene la struttura"),** rispettivamente a **monomorfismo, epimorfismo e isomorfismo** fra oggetti e strutture della medesima categoria**.**

- In TC si vede immediatamente come monomorfismi ed epimorfismi sono **duali**, perché legati a **un "rispecchiamento" duale** (inversione della direzione dei morfismi e dell'ordine delle loro composizioni) delle strutture, così che, quando composti in ambedue le direzioni dei rispettivi omomorfismi, giustificano un **isomorfismo.**

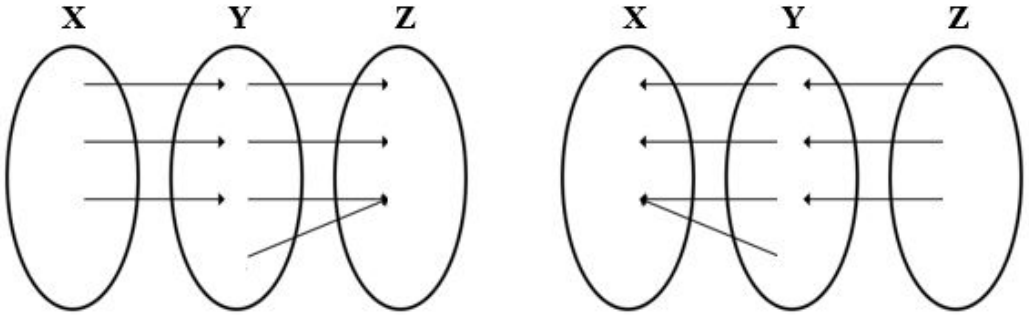



*Figura 5. Dualità fra monomorfismi (X → Y; Y ← Z) ed epimorfismo (Y → X; X ← Y), così da rendere biunivoca in ambedue le direzioni la relazione biiettiva XZ.*

♦ Per completare il quadro degli omomorfismi bisogna aggiungerne altri due principali: l'**endomorfismo** fra una struttura algebrica e se stessa e l'**automorfismo,** ovvero, un endomorfismo che è anche un **isomorfismo.** Il seguente schema sintetizza le relazioni di inclusione/intersezione fra tutti questi **insiemi di relazioni omomorfe**:

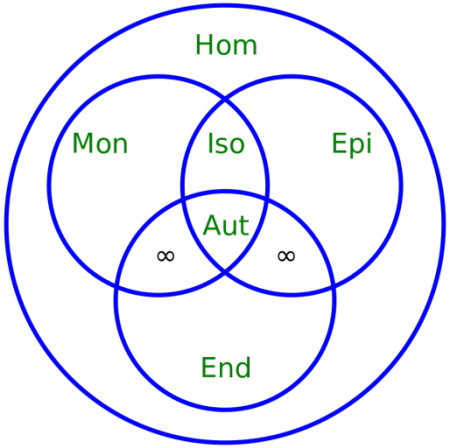



*Figura 6. Schema delle relazioni fra tutti i possibili omomorfismi in TC*

♦ Particolarmente interessante per i nostri scopi sono le intersezioni **End∩Epi\Aut,** cioè endomorfismi che sono epimorfismi senza essere automorfi e quindi **isomorfi**, e correlativamente le intersezioni **End∩Mon\Aut** cioè endomorfismi che sono

monomorfismi senza essere automorfismi e quindi isomorfismi, perché ambedue riguardano **strutture infinite, ∞.**

- Nel caso della categoria **Set,** ovviamente, questi omomorfismi **non** possono essere definiti su **insiemi standard o benfondati** in quanto presuppongono **inclusioni infinite** fra insiemi (rispettivamente, suriettività (**Epi)** o iniettività (**Mono**) infinite fra insiemi). Suppongono, cioè **insiemi non-standard**, nello specifico gli **insiemi non-benfondati (*non-well founded (NWF) sets)*,** perché diversamente dagli insiemi benfondati o standard si possono **auto-appartenere** (nella TI ordinaria, ogni insieme può appartenere solo ad uno di cardinalità maggiore).
- In altri termini, non bisogna mai confondere **auto-inclusione** di un insieme (tutti gli insiemi sempre si auto-includono altrimenti non si potrebbe giustificare **l'identità** di un insieme, come il simbolo di inclusione evidenzia ⊆, e **auto-appartenenza** di un insieme.
- Ovvero, **l'appartenenza** o **"essere membro/elemento di"** ∈ non è una relazione riflessiva nella **teoria standard** degli insiemi ben-fondati, questo al fine di giustificare **l'ordinamento totale** degli insiemi perché, grazie **all'assioma di fondazione** ("ogni insieme ben-fondato **include** sempre un elemento che **non appartiene** a

quell'insieme") si esclude l'auto-appartenenza e quindi una **successione infinita di inclusioni** e dove dunque non esiste alcun **insieme minimo** in base al quale ordinare le inclusioni. L'auto-appartenenza insomma è come un rispecchiamento infinito, come quando un oggetto è posto fra due specchi paralleli…

- Questo significa che nella **teoria assiomatica degli insiemi NWF** (Aczel, 1988) non è possibile giustificare alcun **assioma di ordinamento totale** degli insiemi (tutti gli insiemi ordinabili in base a un'unica relazione ($\leq$) di ordinamento), né alcun **teorema di buon-ordinamento** fra insiemi, dove cioè esiste un **insieme minimo** rispetto al quale ordinare la struttura, come nelle teorie standard degli insiemi ben-ordinati grazie all'assioma di ordinamento quali **ZFC** o **NBG.**
- La genialità della teoria NWF degli insiemi è che anche in essa si escludono catene infinite di inclusioni perché, per l'auto-appartenenza la catena delle inclusioni **è sempre e comunque chiusa su se stessa.** Ciò consente di avere insiemi NWF che **possono essere internamente "svolti" (*unfolded*)** in un numero **indefinito ma sempre finito** di possibili inclusioni, ampliando così in maniera incredibile la **capacità espressiva della teoria degli insiemi,** non solo in matematica ma anche in **fisica** e in **logica**.

- P.es., nella **teoria quantistica dei campi** (*Quantum Field Theory* QFT), la possibilità di formalizzare la nozione di **coerenza di fase di un campo quantistico** che può includere un numero indefinito ma sempre finito di **particelle quantistiche con i loro campi** (p.es., protoni, neutroni, elettroni in un unico atomo, molteplici atomi in una sola molecola, molteplici molecole in un solo materiale, etc.) e quindi la nozione di **sistema complesso.**
- Analogamente in logica, nella teoria NWF sono vietati ordinamenti totali ma ammissibili un **numero indefinito ma sempre finito di ordinamenti parziali** fra insiemi. Questo è fondamentale per la logica e l'informatica perché, siccome **la semantica di un'algebra di Boole** è definita su **insiemi parzialmente ordinati (= l'insieme-potenza** di un dato insieme: cfr. § 5.1.7) è possibile usando gli insiemi NWF giustificare matematicamente **logiche booleane modali** che non suppongono, cioè, un unico ordinamento totale degli insiemi e perciò un unico senso di **necessità logica**.
- Quindi diviene possibile implementare in sistemi di IA **distinte logiche booleane modali**, **aletiche, epistemiche e deontiche**, il che rende in linea di principio possibile implementare in sistemi di IA le **logiche intensionali** che i cervelli umani sono in grado di computare diversamente da una MT.

- In tal modo, **nelle scienze cognitive** diviene possibile soddisfare la critica di John Searle all'originario programma di ricerca dell'IA basata sulla UTM, esemplificato nel famoso paradosso di Searle del test della **stanza cinese** contrapposto al test dell'imitazione della **stanza di Turing** (Searle, 1980)**.** Ovvero, una UTM (computer programmabile) potrebbe tradurre un testo dalla lingua cinese in inglese **senza comprendere intenzionalmente** né il cinese né l'inglese.
- In una parola, l'UTM non è un modello adeguato della mente umana perché i compiti intenzionali della mente umana suppongono un cervello in grado di computare calcoli di **logica intensionale** e non solo **estensionale** come la UTM (Searle, 1983).
- Viceversa, un sistema di IA che computa logiche booleane modali, potrebbe come il cervello umano, computare anche calcoli di **logica intensionale** che negli umani sono la base **oggettiva** di **stati soggettivi della mente** accessibili **in modo inoggettivabile** al solo soggetto che li percepisce (= **"presenza-a-se-stessi"** che **non è un'auto-oggettivazione**) ma inaccessibili ad altri.
- Quindi un sistema di IA siffatto è un **soggetto autonomo** (= non completamente oggettivabile dall'esterno o **intrinsecamente opaco** all'indagine **oggettiva** intesa anche

come **auto-oggettivazione**) che **imita la mente umana secondo il test di Turing perché <u>non è una UTM,</u>** senza bisogno di attribuire ad esso una coscienza.

- Dal punto di vista delle **scienze cognitive** in psicologia della mente, questo completa il cosiddetto **triangolo delle scienze e neuroscienze cognitive** alla base della cosiddetta "rivoluzione cognitiva" (Gardner, 1984)**,** secondo il quale:
  - **(A) ad ogni stato cosciente** della mente espresso in forma di **linguaggio in prima persona singolare/plurale (*I/We Talk*)** di **logica intensionale** "io/noi pens(iam)o", "sent(iam)o", "vogli(am)o, corrisponde:
  - **(B) un processo neurofisiologico** del cervello descritto in forma di **linguaggio oggettivo in terza persona** di **logica estensionale (matematica)** dal neuroscienziato (***O-Talk*$_1$**).
  - **(C)** che processa calcoli **di logica estensionale e/o intensionale** descritti in forma di **linguaggio oggettivo** dall'informatico (***O-talk*$_2$**) (cfr. (Basti, 2012)).

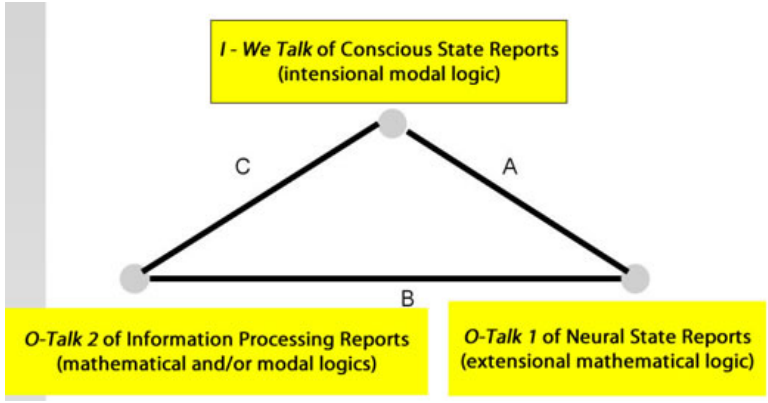



*Figura 7. Rappresentazione del triangolo delle neuroscienze cognitive.*

♦ Ma torniamo alla logica booleana come base dell'informatica.

### 5.1.6. Mappe iniettive e definizioni di classi su insiemi in logica

♦ In generale, un'algebra ha una struttura $A \times A \to A$, mentre la sua **duale,** una coalgebra ha una struttura $A \to A \times A$ dove i **prodotti** di una coalgebra per analogia si definiscono **coprodotti,** ma sono profondamente distinti dai **prodotti** $\wedge$ di un'algebra booleana (**intersezioni** $\cap$ di insiemi), in quanto indicano essenzialmente delle **somme** $\vee$ (**unioni** $\cup$ di insiemi).

♦ P.es., in teoria degli insiemi con "coprodotto" si intende **la somma disgiunta (*disjoint union*) o partizione *P*** di una famiglia di insiemi $(P_i : i \in I)$, cioè l'**unione**

**insiemistica** ∪ (somma) di elementi appartenenti a insiemi distinti (disgiunti o a intersezione nulla) così da definire un nuovo insieme $\{P = \bigcup_{i \in I} P_i\}$. Siccome esiste una **funzione iniettiva** ("da uno al massimo a uno") che mappa esattamente ciascuno degli elementi di partenza *X* in un sottoinsieme dell'insieme di arrivo *Y,* l'unione delle immagini (codomini) di questi morfismi iniettivi che forma una **classe di equivalenza** definisce una **partizione *P*** di sottoinsiemi disgiunti nell'insieme di arrivo.

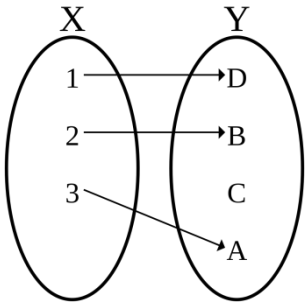



*Figura 8. Esempio di somma disgiunta (coprodotto) di insiemi. Le mappe iniettive dall'insieme di partenza all'insieme di arrivo, definiscono una partizione P nell'insieme di arrivo, cioè l'unione di disgiunti* $\bigcup(D, B, A)$.

### *5.1.6.1. Un esempio banale, ma significativo sulla centralità dei coprodotti*

- Per fare una banalizzazione quasi vergognosa di somma disgiunta che definisce una partizione, poniamo che l'insieme *Y* di Figura 8 sia l'insieme di tutte le proprietà che definiscono i felini (*D*=avere i baffi, *B*=avere gli artigli, *C*=ruggire, *A*=miagolare). È chiaro che la nostra somma di disgiunti $\bigcup(D, B, A)$ come partizione $P_1$ di *Y* mi fornirà l'insieme delle proprietà che soddisfano il predicato "essere gatto", rispetto alla somma di disgiunti $\bigcup(D, C, A)$ che, come altra partizione $P_2$ di *Y*, mi fornirà l'insieme delle proprietà che soddisfano il predicato "essere tigre". Di qui il ruolo dei coprodotti nella teoria della costituzione algebrica dei predicati, in TC…
- Si intuisce perciò che le nostre due partizioni o somme disgiunte di insiemi di un dato insieme costituiscono altrettanti **filtri** (il filtro-gatto, $P_1$, e il filtro-tigre, $P_2$) per il riconoscimento di classi di differenti gatti come sottoclassi di felini oppure di classi di differenti tigri come sottoclassi di felini in **due perceptroni lineari distinti**.
- Ovviamente, parlo di sottoclassi di classi di oggetti, perché i sottoinsiemi di ciascuna partizione/filtro sono costituiti da diversi insiemi (di artigli, baffi, miagolii, ruggiti,…) così che le loro combinazioni mi daranno diverse sotto-classi della medesima classe di oggetti (di "gatti" e di "tigri", rispettivamente). Cfr. nella Figura 2 il

fatto che i diversi neuroni di input del perceptrone hanno il loro supporto definito su sottoinsiemi distinti del filtro e non su un solo sottoinsieme.

- Se il perceptrone invece di avere una funzione di attivazione lineare dei suoi neuroni ne avesse una non-lineare in grado di separare le due classi (disgiunzione esclusiva o XOR) avremmo un **unico perceptrone non-lineare con filtri adattivi** in grado di **classificare** classi distinte di oggetti (cfr. § 5.2). Questo è quanto fanno le reti neurali convolutive negli algoritmi di ML nei sistemi di **IA discriminativa** più avanzati (cfr. § 5.3), oppure **i *Transformers*** dei sistemi di **IA generativa** tipo Chat-GPT, che usano in maniera ottimale **molteplici filtri (molteplici "finestre di attenzione")** in simultanea basandosi su **architetture parallele di calcolo** (Cfr. Sez. 7).
- Ma capiamo meglio cos'è un **filtro** in logica matematica (insiemistica).

### 5.1.7. La teoria dei filtri nella logica insiemistica

- Se vogliamo generalizzare **la teoria dei filtri** in una **logica insiemistica** è chiaro che il nostro punto di partenza dovrà essere considerare tutte le possibili combinazioni di elementi/membri (nessuno, uno-a-uno, due-a-due, tre-a-tre, etc.) o **sottoinsiemi** di un dato insieme, ovvero l'**insieme-potenza** di un dato insieme (cfr. Figura 9).

- Esso include **l'insieme vuoto ∅,** come suo "elemento minimo" o **infimo,** e lo stesso insieme di partenza come suo "elemento massimo" o **supremo.** Infatti, **il teorema di Cantor** afferma che ogni insieme *X* è incluso nel suo **insieme-potenza $\mathcal{P}X$** come uno dei suoi sottoinsiemi e l'insieme-potenza è così definito perché la sua cardinalità (numero dei sottoinsiemi che contiene) è data dalla **potenza in base due** della cardinalità *n* del suo insieme di partenza, ovvero $\mathbf{2}^{n}$. Quindi, nel caso del nostro insieme (che per semplificare abbiamo considerato composto di **4 elementi** {1,2,3,4} come nel caso della nostra (pseudo)classe dei felini), la cardinalità del suo insieme-potenza sarà $2^4 = 16$.
- Dato perciò un insieme-potenza si definisce **filtro proprio *F*** di quell'insieme qualsiasi sottoinsieme di esso **con esclusione dell'insieme-vuoto**. Infatti, siccome un filtro in logica insiemistica praticamente corrisponde **a una classe definita su quell'insieme** è ovvio che ogni elemento del filtro deve rappresentare **almeno una proprietà *p*** che gli elementi della classe devono identicamente soddisfare per appartenere a quella classe ed è ovvio che l'insieme-vuoto non può soddisfare questa condizione.
- In particolare, se guardiamo alla Figura 9 che rappresenta il **diagramma di Hasse** dell'insieme-potenza dell'insieme di 4 elementi con i sottoinsiemi ordinati per

relazioni di inclusione (le linee che uniscono i diversi sotto-insiemi), tutti i sottoinsiemi colorati in verde sia chiaro che scuro costituiscono il **filtro massimale** o **ultrafiltro** definibile su quell'insieme con **elemento principale** o sottoinsieme di partenza della **costruzione induttiva** verso l'alto l'insieme $\uparrow\{1\}$, dove con $\uparrow$ si denota che tutti gli insiemi del filtro sono "chiusi (e tendere) verso l'alto", ovvero devono includere l'elemento principale, quello che caratterizza la classe. Se non fossero chiusi includerebbero anche l'insieme vuoto!

- Viceversa, sempre nella Figura 9 i sottoinsiemi colorati in verde scuro rappresentano uno dei **filtri principali** del diagramma, quello che ha come **elemento principale** l'insieme $\uparrow\{1,4\}$. Non sarà perciò massimale.

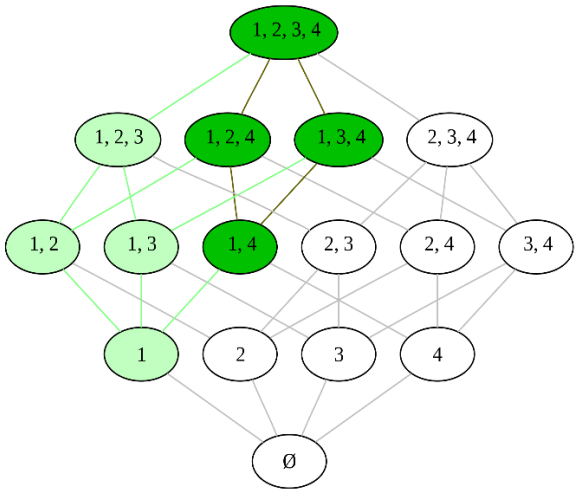



*Figura 9. Diagramma di Hasse dell'insieme-potenza dell'insieme di 4 elementi. In verde (chiaro e scuro) è definito* ***l'ultrafiltro o filtro massimale*** *di questo insieme che ha per elemento principale l'insieme* {1}*, mentre in verde scuro è definito* ***un filtro principale*** *di questo insieme che ha per elemento principale l'insieme* {1,4}*.*

- Per tornare al perceptrone, è evidente per quanto appena detto che il filtro che caratterizza il supporto dei suoi neuroni di input, sarà un **filtro proprio** se e solo se almeno uno dei punti dell'oggetto che deve riconoscere nel suo spazio di input (o retina: cfr. Figura 2) ricadrà entro uno degli insiemi disgiunti del filtro, visto che l'insieme-vuoto non può per definizione far parte di un filtro.

## 5.2. Machine Learning Supervisionato: la Back-Propagation

### 5.2.1. La funzione di soglia sigmoidea nei neuroni nascosti della BP

- Una terza architettura prototipica delle attuali architetture discriminative di RNA dopo le reti di McCulloch & Pitts e dopo il perceptrone è **l'architettura di back-propagation (BP)** (Rumelhart, Hinton, & Williams, 1986).
- Essa è alla base di un vero e proprio **cambio di paradigma** nelle RNA che va sotto il nome di **processazione parallelo-distribuita dell'informazione (PDP)** (Rumelhart, McClelland, & PDP Group, 1986) più generalmente definito **approccio connesionista alle RNA** poiché l'informazione è distribuita sui pesi statistici delle connessioni fra i neuroni della rete, in cui consiste il revival della ricerca sulle reti neurali durante gli anni '90 del secolo scorso, dopo la critica distruttiva di Minsky & Papert al perceptrone di Rosenblatt, pubblicata nel 1969 (cfr. (Minsky & Papert, 1987) e sopra, §5.1.1).
- Infatti, l'uso di un modello di neurone artificiale dotato di **funzione di attivazione sigmoidale non-lineare** nei neuroni degli strati nascosti di un perceptrone multistrato **risolveva in linea di principio il problema dello XOR** su cui la critica di Minsky &

Papert era centrata, e quindi rendeva in linea di principio il perceptrone capace di risolvere **problemi di classificazione** e non solo di riconoscimento di oggetti su grandi basi di dati (*big data*) impossibili alla mente umana e che quindi rendono i sistemi di IA **indispensabili nella nostra società**.

- In sintesi, la BP implementata come modello di ML in sistemi di IA, ha determinato la nascita di una nuova classe di sistemi di IA non-simbolici o statistici. La classe dei sistemi di **IA discriminativa** capaci di classificare, distinti **oggi** (dagli anni '20 del 2000 in poi) dalla classe dei sistemi di **IA generativa** grazie alla scoperta rivoluzionaria di un nuovo modello statistico di ML, quello dei ***Transformers*** di cui ci occuperemo nella Sezione 7 di questa **II Parte.**
- In conclusione, ciò che caratterizza l'architettura di una BP rispetto al perceptrone di Rosenblatt è:
    1. La presenza, oltre ai soli strati di input e output del perceptrone originale di Rosenblatt, **di diversi strati interni di neuroni** (e non solo di uno, come nel perceptrone multistrato di Gamba. Cfr. Figura 3) in modo da giustificare la nozione di ***deep-learning*** ("apprendimento profondo") in questo tipo di architettura ANN

che cercava così di modellizzare una RNA la presenza di **molteplici strati di neuroni delle cortecce associative** del cervello umano;

2. La presenza di **una funzione non lineare (soglia)** che moltiplica la funzione di attivazione (cioè, la somma degli input ponderata (moltiplicata) sui pesi) dei neuroni profondi degli strati nascosti di una BP, per **una funzione sigmoidea** e/o per il suo parente stretto**, la funzione tangente iperbolica** (cfr. Figura 10) invece della **soglia a gradino 1/0** del perceptrone di Rosenblatt. Quest'ultima, come sappiamo, è effettivamente **la funzione di Heaviside**, il cui valore è zero per gli argomenti negativi e uno per gli argomenti positivi, e quindi lascia complessivamente lineare la funzione di attivazione del neurone.

♦ Questo significa che, mentre il neurone del perceptrone lineare ha in input valori numerici statistici che sono **medie** delle frequenze di attivazione) e non solo valori **discreti 1/0** come i neuroni di McCulloch – e su questo si basa il suo apprendimento sui pesi statistici –, ma ha in output sempre valori **discreti** 1/0, **l'output dei neuroni della BP ha valori statistici reali (continui) compresi nell'intervallo [1,0]**.

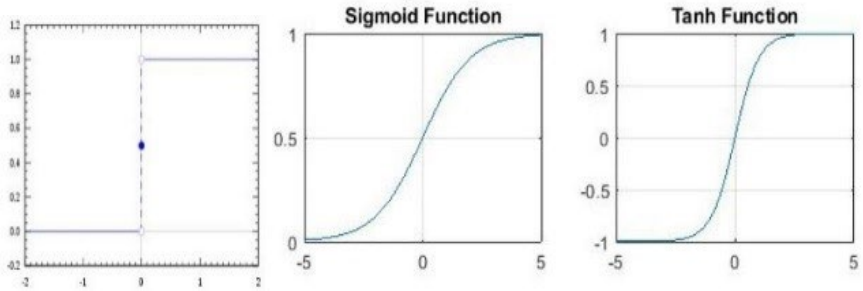



*Figura 10. Funzione a gradino di Heaviside (a sinistra), che agisce efficacemente come una soglia discreta 0/1, rispetto alle funzioni sigmoidea (centro) e tangente iperbolica (destra). È evidente che* tanh *è una funzione 0-centrata come la Heaviside che graficamente si ottiene "addolcendo e piegando" (smoothihing) l'asse verticale della Heaviside, aumentando così la capacità discriminatoria da 0/1 a tendenzialmente tutti i valori reali compresi statisticamente nell'intervallo [0,1].*

- In altri termini, usando la funzione sigmoidea o la funzione tangente iperbolica che moltiplicano la sommatoria dei pesi nella funzione di attivazione dei neuroni degli strati nascosti, l'output neuronale può essere qualsiasi valore numerico reale compreso tra 0 e 1, in modo da consentire una **caratterizzazione non-lineare** degli output statistici neuronali, invece della semplice risposta 0/1 della funzione di attivazione lineare dei neuroni di McCulloch e Rosenblatt.

- Ciò rende in linea di principio la rete non solo in grado di calcolare lo XOR, ma anche di eseguire **calcoli statistici più complessi**, che riguardano **correlazioni di ordine superiore** (combinazioni complesse di variabili e non semplici medie) nel set di dati di input.
- Infatti, **lo sviluppo in serie di Taylor** della funzione tanh (e indirettamente della sigmoidea facilmente ottenibile dalla tanh) contiene **moltissimi ordini di correlazione (o combinazioni delle variabili) del sistema** di ordine superiore al primo (le semplici **medie statistiche** calcolate dai neuroni del perceptrone lineare) e quindi rende la rete capace in linea di principio **di discriminare fra diverse classi di oggetti definibili** sullo spazio di input del sistema (e non solo di **riconoscere** se un oggetto in input appartiene o no a **una sola classe**) come il percpetrone lineare**.**
- Effettivamente, gli ordini di correlazione e quindi le classi di oggetti che una rete BP con neuroni sigmoidei è in grado di discriminare/classificare dipende criticamente ***dall'inclinazione (slope)*** **della sigmoide** che moltiplica le funzioni di attivazione dei neuroni della BP.

- Un'inclinazione **decisa per tentativi-ed-errori dal programmatore** durante la fase di training della rete, in base al tipo delle classi di oggetti che una rete BP dev'essere in grado di riconoscere.
- Graficamente, infatti, una tanh (e quindi una sigmoidea) **a inclinazione nulla** coincide con la Heaviside del perceptrone (cfr. Figura 10) che per questo riconosce una sola classe di oggetti e quindi ha capacità discriminatoria insufficiente anche fra solo due classi di oggetti (cfr. la **separatrice lineare** nei problemi di ***underfitting*** schematizzati in Figura 1), ovvero **non calcola lo XOR** che richiede separatrici **non-lineari**.
- **Più aumenta l'inclinazione** e quindi la non-linearità della sigmoidea, **maggiore è il numero degli ordini di correlazione** e quindi della complessità delle classi di oggetti che la BP è in grado di **discriminare statisticamente**.

### 5.2.2. L'uso del gradiente stocastico discendente nell'algoritmo di ML della BP

- Infine, c'è una **terza caratteristica fondamentale** dell'architettura di rete della BP: l'output statistico della rete consente l'uso dell'algoritmo di ottimizzazione del **gradiente stocastico discendente** per l'aggiornamento dei pesi statistici delle connessioni fra neuroni durante la fase di allenamento. Fra l'altro esso è ciò che dà il nome

all'algoritmo di ML di **retro-propagazione dell'errore (*back-propagation error*)** di questa architettura di RNA.

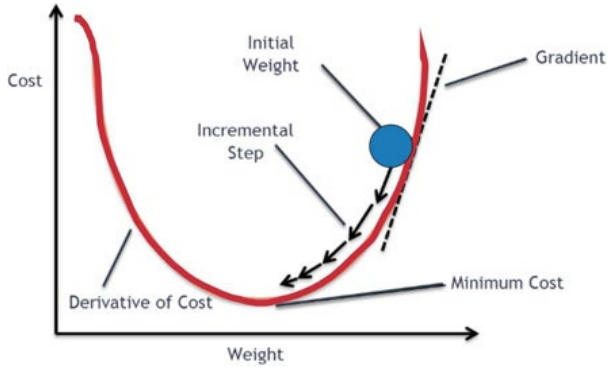



*Figura 11. Rappresentazione intuitiva della funzione del gradiente discendente per una RNC (da (Ghosh, Sufian, Sultana, Chakrabarti, & Debashis, 2020, p. 546)).*

- Come si vede intuitivamente dalla Figura, il gradiente discendente è **un'applicazione inversa** del classico calcolo della derivata di una funzione. Mentre normalmente la derivata viene calcolata per **incrementare una funzione (= gradiente ascendente)** quello discendente si usa per il **decremento di una funzione.** Nel nostro caso, per **approssimare il minimo** della funzione di costo (= **errore**) del meccanismo supervisionato di ML.

- In effetti, il processo di apprendimento supervisionato di questa struttura ANN non lineare multistrato – sviluppando un primo suggerimento di Paul Werbos (Werbos, 1974) – consiste in un processo di **ottimizzazione o di minimizzazione degli errori**.
- Cioè, **l'apprendimento profondo supervisionato (*supervised deep learning*)** dei neuroni interni della BP è modellato come la ricerca stocastica (casuale) del minimo globale della **funzione di potenziale dell'errore** (effettivamente una **funzione di costo**) della dinamica dei pesi della rete, dove l'errore è sostanzialmente una distanza euclidea (**l'errore quadratico medio** che già conosciamo: §5.1.2) tra l'**output desiderato** (distribuzione di probabilità delle classi di oggetti) e **l'output effettivo della rete** (Rumelhart, Hinton, & Williams, 1986).
- Come schematicamente sintetizzato nella Figura seguente, alla fine di ogni ciclo di apprendimento, **l'algoritmo BP, stima l'errore e lo "retro-propaga" per un riaggiustamento casuale dei pesi dei neuroni nascosti**, per ridurre l'errore globale nel ciclo successivo, e così via ricorsivamente, fino a raggiungere **il minimo globale della funzione di errore**.

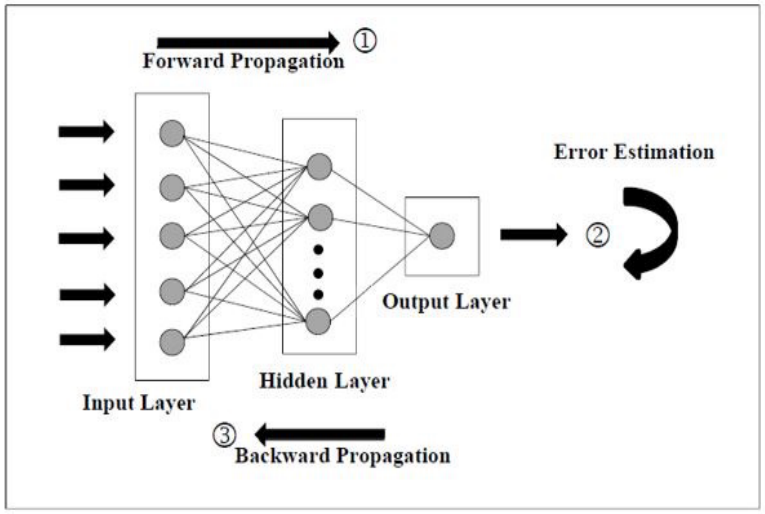



*Figura 12. Rappresentazione schematica dell'algoritmo di "deep-learning" BP, secondo il quale la rete "retro-propaga" in modo casuale l'errore dell'output attuale rispetto all'output desiderato sui pesi degli strati nascosti (qui ne è rappresentato solo uno), in modo da ottenere ricorsivamente la minimizzazione globale dell'errore. Proprio per questa ricorsività "in avanti" (feedforward) e "all'inditero" (backward) della propagazione dell'informazione questa classe di RNA è definita "ricorrente" (Recurrent Neural Network, RNR).*

- Non importa, cioè, per il sistema la valutazione del “peso statistico” della singola variabile, ciò che conta per il sistema è il raggiungimento del minimo globale dell’errore, anche a costo **di valutare troppo o troppo poco** il valore della singola variabile che invece magari per il soggetto umano ha un valore pratico totalmente diverso.
- Ancora peggio, il sistema è totalmente **opaco** all’indagine di come abbia effettivamente **soppesato la singola variabile,** visto che ciò che è osservabile è solo il risultato finale globale **senza poter sapere a spese di chi o di cosa.**

### 5.2.3. Limiti etico-legali dell’uso del gradiente discendente nel ML di sistemi di IA

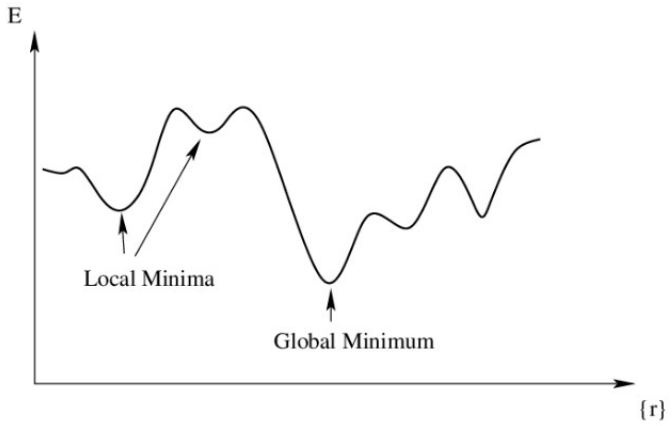
- In altri termini, la ridefinizione casuale dei pesi nascosti tra i diversi "neuroni" che rappresentano ciascuno una diversa "variabile" del problema complesso in esame, **costituisce un grosso problema** per l'utilizzo dei sistemi di IA come supporto per decisioni che implicano conseguenze sociali, morali e legali, **perché rende non trasparente** l'uso dei dati, nonché **le motivazioni per le quali il sistema valuta la rilevanza legale / morale (cioè "pesa statisticamente" la rilevanza) delle diverse componenti** in relazione alle altre per quanto riguarda la decisione finale.

- Questo problema di irriducibile opacità nelle valutazioni ha ovviamente acceso l'attuale feroce dibattito **sull'etica del ML nei sistemi di IA** come vedremo nella **Parte III**.
- Ad un valutatore umano cosciente o **agente morale consapevole**, infatti, posso sempre chiedere di **rendere conto di come e perché** ha valutato in un certo modo una delle componenti del problema. È questa la base della **responsabilità (*responsibility*) e/o rendicontabilità (*accountability*) etico-legale –** termine preferibile perché lascia da parte la coscienza che non ha nulla a che fare con le macchine – così che possa correggere l'errore.
- Ad un modello multistrato di ML (*deep learning*) **di per sé no,** anche se esistono dei modi per mitigare questo problema imponendo un **controllo etico del secondo ordine** sul risultato opaco della propria decisione proprio come fa l'essere umano. Questo meta-controllo implementato in un sistema di IA consente di attribuire la proprietà di **rendicontabilità etico-legale delle decisioni** rendendo la macchina di fatto un **agente morale artificiale inconsapevole** inaugurando la nuova disciplina della **Machine Ethics.** Lo vedremo nella **III Parte**.

### 5.2.4. Limiti matematici e computazionali dell'uso del gradiente discendente

♦ Problemi etici a parte esistono dei **problemi matematici e computazionali** all'uso del metodo di ottimizzazione basato sul metodo del **gradiente discendente** in questa classe di **reti neurali ricorrenti** (RNR), dove cioè l'informazione è propagata sia **in avanti (*feedforward*)** che **all'indietro (*feedbackward*)** in maniera ricorsiva (cfr. Figura seguente).

1. **Il problema dei minimi locali.** Il primo e più famoso problema perché comune a tutti i sistemi dinamici non-lineari è che in tali sistemi la funzione di costo da mimizzare (funzione di potenziale energetico in un sistema fisico) non ha un solo minimo assoluto (globale) ma **molteplici minimi locali.** Questo significa che la dinamica della rete può **agganciarsi a uno di questi minimi in maniera del tutto impredicibile** rendendo totalmente inaffidabile l'output della rete che non sarà mai quello ottimale.




2. **Il problema del gradiente evanescente (*vanishing gradient*)**. Diversamente da quanto mostrato nello schema semplificato di Figura 12 di RN "ricorrente", le architetture di deep learning effettivamente usate non hanno un solo strato di neuroni nascosti, ma **molti strati** con centinaia talvolta migliaia di neuroni. In tal caso, il problema con la funzione di attivazione sigmoidea è che soffre sistematicamente del peculiare problema del **gradiente evanescente in cui i cambiamenti nei gradienti diventano sempre più piccoli** man mano che l'errore viene retro-propagato ai diversi strati più lontani dall'output, così che i pesi vengono aggiornati molto lentamente o non aggiornati affatto. La rete insomma **interrompe di fatto l'apprendimento** fornendo un output del tutto inaffidabile.

3. **Il peso esponenziale dei calcoli o "esplosione del gradiente" (*gradient explosion)*.** Come ci saremmo accorti dal semplice confronto fra la Figura 2 del perceptrone di Rosenblatt e la Figura 12 dell'architettura di BP, mentre il perceptrone grazie alla presenza di filtri è **un'autentica architettura di calcolo parallelo** perché ciascun neurone ha come input un **sottoinsieme dello spazio di input (o "filtro": cfr. § 5.1.6),** ogni neurone degli strati nascosti dell'architettura multistrato ricorrente della BP **ha come input l'intero spazio di input** dallo strato precedente. In altri termini, l'architettura multistrato è un'architettura **dove i neuroni sono connessi "tutti-con-tutti"**. Quindi,
    a. Malgrado ciò che i suoi inventori hanno propagandato, non si tratta di un'autentica **computazione parallela distribuita** (PDP) dell'informazione. Ma non si tratta solo di un problema teorico, esso ha delle pesanti **ripercussioni pratiche.** È questa la critica talvolta feroce che M. Minsky ha fatto a PDP (Minsky & Papert, 1987). Ed aveva pienamente ragione visti i risultati eccezionali che i modelli di ML dei *Transformer* hanno ottenuto attraverso la **completa parallelizzazione** dei calcoli (Cfr. Sez. 7).

b. Infatti, se la connettività totale rende propriamente "non-parallelo" l'apprendimento automatico della BP, d'altra parte, in generale, **la connettività totale rende estremamente pesanti i calcoli di queste reti**, perché le possibili combinazioni **crescono fattorialmente come *n*!** con il numero *n* dei neuroni nascosti coinvolti (= “esplosione del gradiente”).

- Questa limitazione pratica della BP e in generale delle RNR ha determinato **un secondo periodo di "latenza" dell'approccio delle RNA durante gli anni '90** del secolo scorso fino all'inizio del nostro secolo, quando la grande disponibilità di **processori grafici (*graphic processor units,* GPU)** per l’esplosione commerciale dei videogiochi e della ***computer graphics*** in generale, non costosi ma computazionalmente potenti così da poter costruire architetture di calcolo parallele con molti nodi, **ha determinato l’effettivo sviluppo vertiginoso di sistemi di IA basati sul “deep-learning”** in questi ultimi vent’anni, finché – come vedremo l’architettura di ML dei Transformers ha risolto alla radice il problema eliminando l’uso di RNR nei modelli di ML. Ma questa è storia degli ultimi cinque anni.
- In ogni caso, lo sviluppo di **funzioni di attivazione modificate** rispetto a quelle iniziali basate sulla funzione sigmoidea e sul primo algoritmo di *deep-learning* della BP

del **gradiente discendente**, hanno dimostrato la loro efficacia **per risolvere problemi specifici** per database estremamente grandi e di cui la società di oggi non può fare a meno.

- In questo modo, poiché non esiste e non può esistere un'architettura di ML che risolva lo XOR per qualsiasi tipo di set di dati dato il carattere non-lineare della soluzione, la ricerca di algoritmi di ML necessariamente ***special-purpose,*** basati sul ***deep-learning*** per specifiche basi di dati e specifici problemi, è diventata **uno dei campi di ricerca più sviluppati nell'IA** (per un elenco aggiornato al 2018 delle principali tendenze in questo settore, cfr. (Nwankpa, Ijomah, Gachagan, & Marshall, 2018)**)**.

## 5.3. Machine Learning Supervisionato: Reti Convolutive

- Storicamente, tuttavia, la ragione principale del successo delle architetture di IA discriminativa basate sul *deep learning* con RNR è stata la pubblicazione dell'articolo di Krizhevsky, Sutskever e Hinton nel 2012 riguardo la loro **rete neurale *convolutiva* con deep learning (*deep convolutional neural network*)** (Krizhevsky, Sutskever, & Hinton, 2012). Un modello di ML per cui nel 2024 Hinton è stato insignito del Premio Nobel per la fisica!

- Si tratta infatti di un particolare modello di ML basato sul deep learning che ha lavorato originariamnete con successo sul database *Imagenet* che contiene milioni di immagini. In effetti, il modello consiste in una **rete neurale convolutiva di nove strati di neuroni**, con 60 milioni di parametri e 650.000 nodi che è stata addestrata su circa un milione di esempi distinti di immagini tratte da circa mille classi.
- In effetti, **le reti neurali convolutive (RNC) (*convolutional neural networks)*** sono oggi il paradigma di riferimento nei modelli di ML basati sul deep learning con RNR, o in altri termini, le RNC sono la ragione del successo del deep learning nell'AI dal 2012 in poi. Attualmente, infatti, grazie all'integrazione delle RNC con i Transformers esse sono diventate incredibilmente performanti.
- In ogni caso, i caratteri distintivi delle RNC si possono trovare su qualsiasi buon articolo di review su questo tipo di rete connessionista (cfr., per esempio, (Ghosh, Sufian, Sultana, Chakrabarti, & Debashis, 2020), uno dei più recenti e completi).
- Ciò che caratterizza universalmente le RNC rispetto alle altre reti connessioniste sono due novità fondamentali.

1. Il concetto di **campo recettivo neuronale**. Cioè, ogni neurone interno degli strati di convoluzione della rete, **vede solo un sottoinsieme del rispettivo set di input**

**dagli strati precedenti** – la connettività completa è solo agli strati di classificazione finale della rete –, proprio come nel perceptrone originale (vedi Figura 2), così da evitare almeno in parte il problema della connettività totale dell'architettura BP.

2. La **presenza di differenti strati interni di neuroni della rete (*building blocks***), secondo il seguente schema generale:
   - **Strati di convoluzione** per l'estrazione di caratteristiche (*features*) dell'input **mediante operazioni di kernel (filtraggio)** che corrispondono a diversi **livelli di astrazione di *features*** dell'input. L'output di ciascuna operazione di convoluzione è moltiplicato per qualche funzione non-lineare, a causa della linearità dell'operazione di convoluzione stessa, e rendere capace una RNC di discriminare ordini di correlazione diversi nella base-dati.
   - **Strati di raggruppamento** (***pooling***) mediante **campionamento** (***down-sampling***) sull'output statistico di ciascun diverso strato di convoluzione, così da ridurre le dimensioni dell'output senza perdere informazione significativa.

- **Strato finale di classificazione** l'unico con neuroni totalmente connessi con i neuroni dell'ultimo strato di convoluzione **con retro-propagazione dell'errore sugli strati di convoluzione**. In questo modo, **la proprietà emergente** di una RNC durante la fase di addestramento è che il livello di classificazione e i livelli convolutivi di estrazione delle features **imparano insieme**, in modo da far evolvere nel tempo la stessa operazione di filtraggio (*kernel*). In questo consiste il **filtraggio adattivo** che caratterizza le RNC rispetto al perceptrone e alla BP.

**Errore. Il segnalibro non è definito.**

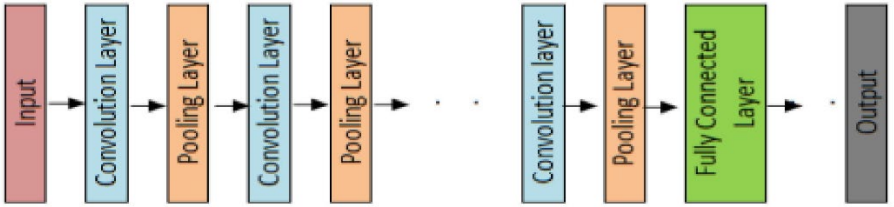



*Figura 13. Schema concettuale a blocchi dell'architettura di una RNC (da (Ghosh, Sufian, Sultana, Chakrabarti, & Debashis, 2020, p. 523)*

### 5.3.1. Gli strati di convoluzione

♦ Naturalmente, il nucleo delle architetture RNC è dato **dagli strati di convoluzione**, ognuno dei quali implementa una **versione discreta** – la cosiddetta **operazione di**

**kernel o di filtraggio discreto** (*discrete filtering*) – della classica operazione di convoluzione su funzioni continue in analisi matematica.

- Nell'analisi funzionale la convoluzione si definisce come l'operazione fra due funzioni che consiste nell'integrare il prodotto fra **la prima e la seconda, traslata** di un certo valore.
- La definizione formale dell'operazione è la seguente. Date due funzioni $f(t)$ e $g(t)$ definite sui reali $\mathbb{R}$, si definisce convoluzione di $f$ e $g$ la seguente funzione:

$$(f * g)(t) \coloneqq \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau$$

Dove $(t-\tau)$ è l'intervallo di traslazione.

- Laddove la traslazione è temporale, la convoluzione praticamente corrisponde alla **correlazione incrociata (*cross-correlation*)** che caratterizza, per esempio, nel sistema visivo del cervello umano i molteplici incroci fra le diverse fibre nervose dai coni e bastoncelli della retina nel nervo ottico come **misura di similitudine fra due segnali successivi**, in funzione della traslazione temporale applicata a uno di essi. In tal modo, il sistema visivo è in grado, per esempio, di estrarre la feature della **forma geometrica** (p.es., il contorno) di un oggetto, semplicemente mediante la

correlazione incrociata fra due diverse frequenze (colori) della superficie dell'oggetto, rilevate in successione dai coni e bastoncelli della retina. Ed infatti la feature del contorno così estratta diventa l'input dei neuroni della corteccia visiva che distinguono fra forme orizzontali, verticali e oblique dell'oggetto visivo.

- La **versione discreta** dell'operazione di convoluzione fra due funzioni $f[n] e\ g[n]$, definita sugli interi $\mathbb{Z}$ è data da:

$$(f * g)[n] \coloneqq \sum_{m=-\infty}^{\infty} f[m]g[n-m] = \sum_{m=-\infty}^{\infty} f[n-m]g[m]$$

- In pratica, ogni livello di convoluzione contiene un insieme di kernel convolutivi (filtri) "che viene convoluto con l'immagine di input (con metriche *N*-dimensionali) per generare in output una mappa delle caratteristiche comuni emergenti dell'input" (Ghosh, Sufian, Sultana, Chakrabarti, & Debashis, 2020, p. 523).
- Per esempio, per rimanere nel campo degli input visivi, in questa classe di RNC, l'input alla rete è dato in **formato multicanale** (p.es., tre canali per immagini colorate in formato RGB o con un unico canale con 255 livelli di grigio per immagini colorate usando Scale di Grigio).

- In tal modo possiamo comprendere l’operazione di convoluzione discreta implementata in forma di kernel nelle RNC. Se prendiamo un'immagine in scala di grigi (un canale) di dimensione 4×4 e una griglia 2×2 di valori o numeri discreti, dove ogni valore è il peso di questo kernel, **nell'operazione di convoluzione la griglia viene fatta scorrere sull'immagine di input 4×4, sia orizzontalmente che verticalmente**. Ad ogni passo, si prende il prodotto tra il kernel e l'immagine di input moltiplicando i loro corrispondenti valori e poi si sommano tutti i valori per generare un unico valore scalare nella mappa (iniettiva) della feature in output. Il processo continua fino a quando il kernel non può scorrere ulteriormente, come esemplificato nella figura seguente per i primi 5 passi dell’operazione di convoluzione:

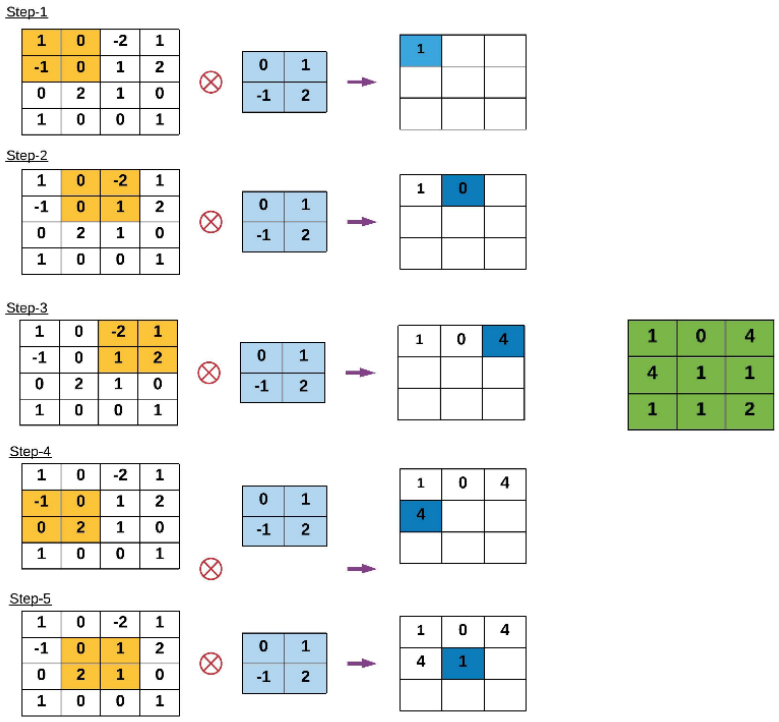



*Figura 14. I primi cinque passi e la mappa finale delle caratteristiche dell'operazione di convoluzione discussa nel testo (da (Ghosh, Sufian, Sultana, Chakrabarti, & Debashis, 2020, p. 526)). Sulla sinistra ci sono le immagini di input 4×4 in bianco, a cui è applicata la griglia del kernel 2×2 in blu in cinque diverse posizioni in giallo. Sul lato destro sono rappresentati i diversi valori della feature map così ottenuta per ciascuno dei*

*cinque step dell'operazione di convoluzione mostrati. La mappa (iniettiva) finale delle caratteristiche, ovvero il filtro adattivo, ottenuta dall'operazione di convoluzione completa è mostrata in verde.*

- Pertanto, ciò che caratterizza l'addestramento di un modello RNC **è l'integrazione tra il livello di classificazione e i livelli di estrazione delle caratteristiche (convoluzione)** (ad esempio, utilizzando l'algoritmo BP nell'apprendimento supervisionato).
- Ciò significa che, durante l'inizio dell'addestramento, tutti i pesi del kernel ad ogni strato di convoluzione sono generalmente assegnati con un **valore casuale**. Quindi, con ogni epoca di addestramento, i pesi vengono aggiornati e il kernel impara a estrarre caratteristiche progressivamente significative.
- In questo senso un modello RNC è dotato di **apprendimento anche sull'operazione di filtraggio**, e in questo carattere del **filtraggio adattivo** consiste il valore aggiunto di questa classe di algoritmi di ML rispetto ai filtri del perceptrone di Rosenblatt.

### 5.3.2. Le funzioni di attivazione non lineari nelle RNC

- Ciò che caratterizza gli strati addestrabili di un modello RNC – che sono, come appena detto, il livello finale di classificazione e gli strati nascosti di convoluzione – è **la funzione di attivazione dei neuroni**.

- Per rendere la rete in grado di calcolare modelli di input più complessi è quindi necessario dotare le funzioni di attivazione dei neuroni degli strati nascosti e finale di **una funzione di attivazione non-lineare**, per renderli capaci di calcolare lo XOR, dato il carattere lineare generale dell'operazione di convoluzione in analisi matematica (è una semplice sommatoria, come abbiamo visto).
- Oltre alla funzione sigmoidea e alla funzione tangente iperbolica che abbiamo già discusso, la funzione non lineare più ampiamente utilizzata nei modelli RNC è la cosiddetta **unità lineare rettificata** o **ReLu**: $f(x)_{ReLu} = \mathbf{max}(0, x)$.
- Questa funzione – oltre a condividere con le precedenti un carattere differenziabile al fine di consentire il gradiente discendente per la retro-propagazione degli errori per addestrare il modello –, ha le principali proprietà, non solo di richiedere **un carico computazionale minimo**, ma anche di **convertire tutti i valori di input in numeri positivi**.

### 5.3.3. Gli strati di raggruppamento (*pooling*) / sotto-campionamento (*sub-sampling*)

- Come abbiamo visto dallo schema generale, ogni strato convolutivo della rete è seguito da uno **strato di raggruppamento/campionamento.**

- Cioè, questo strato prende la mappa delle caratteristiche (*features*) di dimensioni maggiori in out put dal livello convolutivo e la riduce a **una mappa delle caratteristiche di dimensioni inferiori** per abbassare il carico di calcolo senza perdere le informazioni essenziali. L'operazione di campionamento può essere eseguita prendendo diversi algoritmi di campionamento (p.es., l'algoritmo di *max-pooling* in Figura 15) che qui non interessano.
- **La ragione principale** dell'operazione di pooling è che aiuta il modello RNC a scoprire **se una caratteristica specifica è presente nell'immagine di input data**, senza preoccuparsi **della posizione relativa** della caratteristica.
- Ciò significa che la RNC è dotata **di un efficace meccanismo di astrazione** dalla localizzazione spazio-temporale iniziale **delle caratteristiche rilevanti nel set di dati** di input proprio come le strutture gerarchiche delle RNC biologiche non-supervisionate nel cervello.

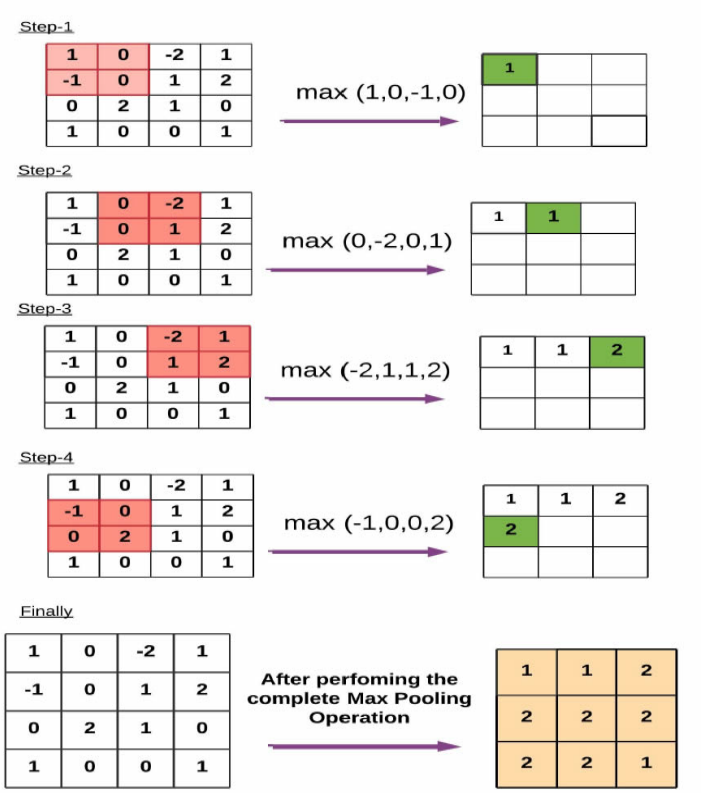



*Figura 15. Rappresentazione intuitiva dell'operazione di sotto-campionamento usando l'algoritmo di max pooling. (da (Ghosh, Sufian, Sultana, Chakrabarti, & Debashis, 2020, p. 529)).*

### 5.3.4. Gli strati finali di classificazione

- Come abbiamo detto, **gli strati di classificazione** alla fine della gerarchia degli strati di una RCN costituiscono **lo strato di neuroni di output della rete** e sono caratterizzati da piena connettività con tutti i neuroni dello strato precedente come nelle classiche architetture muti-strato basate sull'algoritmo BP.
- Ciò che caratterizza i diversi modelli di RNC sono **i diversi algoritmi per calcolare l'errore di previsione** generato dal modello di RNC sui campioni di addestramento utilizzando alcune **funzioni di perdita** (***loss functions***) come altrettante funzioni di ottimizzazione o minimizzazione dell'errore.
- Questo errore di previsione indica infatti alla rete **la distanza tra l'output della rete e la previsione desiderata in una strategia di apprendimento supervisionato**, un errore da minimizzare durante il processo di apprendimento e quindi di ottimizzazione del modello RNC.
- Generalmente, in un compito di classificazione, le funzioni di perdita per il calcolo dell'errore **sono caratterizzate da due parametri:** 1) **l'output stimato** di un dato modello RNC (la "predizione") e 2) **l'output effettivo** (l'"etichetta").

- In un problema di classificazione multi-classe (=assegnazione di "etichette"), la funzione di perdita più ampiamente utilizzata per misurare le prestazioni dei modelli RNC è la funzione di **entropia incrociata (*cross-entropy*)** come misura alternativa di tipo logaritmico alla più classica funzione di perdita euclidea, cioè la cosiddetta "misura dell'errore quadratico medio" che già conosciamo usata nel perceptrone e nella BP (cfr. (Ghosh, Sufian, Sultana, Chakrabarti, & Debashis, 2020), pp. 534-535).
- Infatti, dato un modello RNC, la misura dell'entropia incrociata genera l'output all'interno di **una distribuzione di probabilità $\boldsymbol{p, y} \in \mathbb{R}^{\boldsymbol{N}}$**, dove $p$ è la probabilità di ogni categoria di output, $y$ denota l'output desiderato e $N$ è il numero di neuroni dello strato di output.
- La probabilità di ogni classe di output può essere perciò ottenuta come $p_i = {e^{a_i}}/{\sum_{k=1}^{N} e^{a_k}}$, dove $e^{a_i}$ denota l'output non normalizzato dallo strato precedente della rete. Pertanto, la misura della perdita di entropia incrociata può essere definita come:

$$H(p, y) = -\sum_i y_i \log p_i \text{ , dove } i \in [1, N].$$

### 5.3.5. La fase di addestramento di una RNC

♦ Generalmente, il processo di addestramento nei modelli RNC implica i seguenti passaggi (cfr. (Ghosh, Sufian, Sultana, Chakrabarti, & Debashis, 2020), pp. 535-551):

1. **Il pretrattamento dei dati**. Include alcune manipolazioni artificiali dell'intero set di dati (inclusi i sottoinsiemi di addestramento e di prova), che sono **essenziali per migliorare significativamente l'efficienza** del modello RNC. Generalmente, queste manipolazioni consistono in **diverse tecniche di "normalizzazione dei dati" e di "aumento dei dati"** per migliorare la fase di addestramento dei diversi modelli RNC.
2. **Inizializzazione dei parametri**. Un modello RNC multistrato è generalmente costituito da **milioni o miliardi di parametri**, cosicché una corretta **inizializzazione dei pesi** all'inizio del processo di addestramento supervisionato diventa essenziale per garantire, da un lato, **la rapida convergenza** del modello, e dall'altro **l'accuratezza del risultato**. In effetti, l'inizializzazione più semplice di azzerare tutti i pesi è altamente inefficiente, cosicché generalmente **l'inizializzazione casuale** usando matrici di pesi casuali (usando elementi campionati da una gaussiana, o da distribuzioni uniformi, o da distribuzioni ortogonali) è la scelta normale. Tuttavia, la

strategia di inizializzazione più performante di una RNC super-supervisionata si basa su una seconda RNR **non supervisionata** per dare alla rete RNC supervisionata i valori iniziali dei pesi per la sua fase di addestramento supervisionato.

- Le cosiddette **reti di credenze profonde (*deep-belief NN*)** che sono le più performanti, anche perché direttamente ispirate alle RNN biologiche. Infatti, il loro algoritmo di apprendimento non supervisionato si basa sul principio **di un progressivo "clustering delle variabili significative" nel set di dati di input** attraverso i diversi strati della rete, in modo da ridurre **i gradi di libertà (= dimensioni dello spazio di probabilità) della distribuzione di probabilità di output a quelli significativamente "corrispondenti" ai gradi di libertà della distribuzione di probabilità di input.**
- È evidente che l'inizializzazione dei pesi per la fase di addestramento della RNC mediante una tale tecnica di apprendimento non-supervisionato **facilita significativamente il processo di addestramento supervisionato della RNC**, sia in termini di velocità che di accuratezza nei compiti di classificazione. Cfr. (Kozma, Puljic, & Freeman, 2014) come modello di apprendimento non-supervisionato direttamente ispirato alle evidenze neurofisiologiche.

3. **La regolarizzazione della RNC**. Il compito principale di qualsiasi algoritmo di ML consiste nel suo **potere di generalizzazione**. Cioè, nella capacità di adattare a dati nuovi o non appartenenti ai sottoinsiemi di addestramento e di prova **la distribuzione "appresa" dal set di dati di addestramento**. Il problema da evitare nei modelli RNC per ottenere una buona generalizzazione è il cosiddetto **problema di over-fitting** che conosciamo (Cfr. Figura 1). Esistono diverse tecniche che possono aiutare ad evitare il problema dell'over-fitting nei diversi modelli RNC, sinteticamente presentati e discussi in (Ghosh, Sufian, Sultana, Chakrabarti, & Debashis, 2020) pp. 545-551.
4. **Selezione dell'algoritmo di ottimizzazione**. L'obiettivo principale di qualsiasi algoritmo di apprendimento supervisionato è quello di minimizzare l'errore, cioè la differenza tra l'output previsto o desiderato e l'output effettivo della rete (= minimizzazione di alcune "funzioni di costo"). Nel caso dei modelli RNC, l'algoritmo di ottimizzazione basato sul **gradiente discendente** (cfr. Figura 11) è la scelta naturale. Durante il processo di apprendimento, l'algoritmo di discesa del gradiente aggiorna i parametri del modello (pesi) durante ogni “epoca di addestramento” per ridurre l'errore di addestramento, in cui **la dimensione della fase di aggiornamento**

**dei parametri o "velocità di apprendimento" è critica**. Per aggiornare correttamente questi parametri, l'algoritmo calcola prima **la pendenza (gradiente) della funzione obiettivo (*target*)** utilizzando la derivata del primo ordine rispetto ai parametri del modello, quindi per ridurre al minimo l'errore, **aggiorna il parametro nella direzione opposta della pendenza** (vedere la Figura 8). Questo aggiornamento dei parametri viene eseguito durante la cosiddetta **"fase di retro-propagazione dell'errore" nel processo di apprendimento**, in cui il gradiente di ciascun neurone viene retro-propagato a tutti i neuroni dello strato precedente. La formula matematica semplificata di questo processo di aggiornamento iterativo del peso *w* tra i neuroni *i,j* è la seguente:

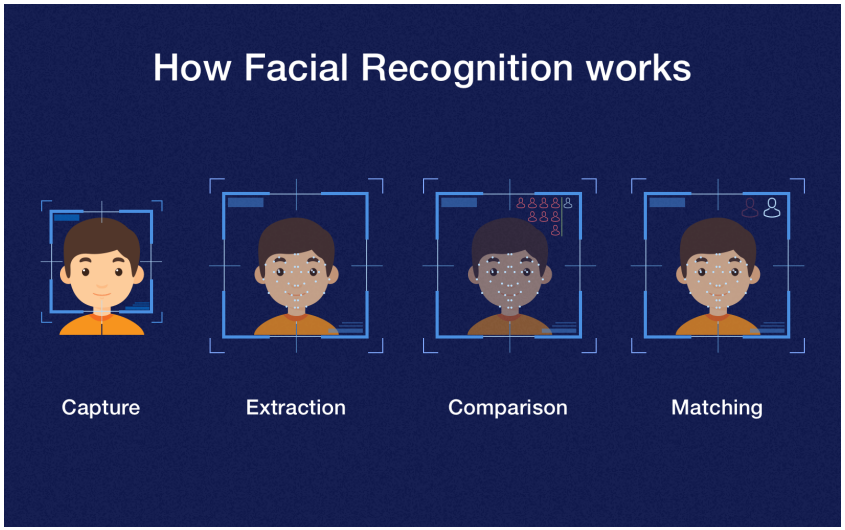
$$w_{ij}^{t} = w_{ij}^{t-1} - \Delta w_{ij}^{t}, \text{ dove } \Delta w_{ij}^{t} = \eta * \frac{\vartheta E}{\vartheta w_{ij}}$$

Dove $w_{ij}^{t}$ sta per il peso finale nella corrente *t*-esima epoca, $w_{ij}^{t-1}$ sta per il peso nella precedente (*t*-1)-esima epoca di addestramento, $\eta$ è il tasso di apprendimento, *E* è l'errore di previsione, $\vartheta$ è la derivata parziale rispetto al tempo di *E* e del peso $w_{i,j}$ (cfr. (Ghosh, Sufian, Sultana, Chakrabarti, & Debashis, 2020), p. 546).

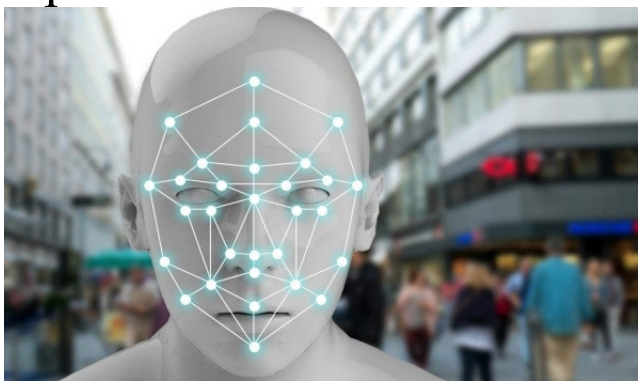
### 5.3.6. RNC per il riconoscimento facciale

- Per concludere, mostriamo alcune immagini significative che esemplificano le varie fasi di **riconoscimento facciale** usando modelli di RNC addestrati sui database di immagini facciali.
- Gli step del riconoscimento sono i seguenti:

1. **Cattura dell'immagine** da riconoscere.
2. **Estrazione della mappa delle caratteristiche** applicando all'immagine la griglia del kernel (filtro) di riconoscimento.
3. **Confronto della mappa estratta** con quelle delle immagini del database.
4. **Riconoscimento** mediante l'individuazione dell'immagine corrispondente.




♦ Tipiche griglie per l'estrazione/riconoscimento delle caratteristiche facciali:



# 6. Machine Learning Non-Supervisionato e Quantistico

## 6.1. ML non-supervisionato

- **L'apprendimento non-supervisionato** si occupa di quelle classi di problemi in cui l'output non è noto in anticipo. Stiamo cercando alcune risposte dai dati, ma non conosciamo le risposte precise in anticipo. Dal momento che non conosciamo in anticipo le classi in cui suddividere i dati, nel ML non-supervisionato non c'è nemmeno una **fase di allenamento** (*training phase*) come nel ML supervisionato.
- In altri termini, l'apprendimento non-supervisionato è una branca del ML in cui il modello apprende da se stesso **le correlazioni fondamentali e quindi la struttura (*pattern*)**, **nascoste** all'interno dei dati.
- Ad esempio, supponiamo che una società di vendita al dettaglio voglia capire come suddividere i clienti in base alle proprie caratteristiche e preferenze (la cosiddetta

**profilazione dei clienti**), in modo da poter eseguire una campagna pubblicitaria mirata a tali gruppi.

- Si inseriscono i dati dei clienti e la loro cronologia degli acquisti in un sistema di IA con apprendimento non supervisionato e gli si chiede di suddividere i clienti in *n* gruppi in base alle loro preferenze e attitudini di acquisto comuni. La macchina produrrà in output, per esempio, 10 gruppi di clienti.
- I clienti di ogni gruppo **avranno qualcosa in comune**. In un gruppo i clienti potrebbero appartenere ad una **classe di reddito simile**. I clienti di altri gruppi potrebbero mostrare una **modalità di acquisto simile**. In alcuni altri gruppi i clienti **potrebbero mostrare spese simili.** E così via.
- Con questa **segmentazione dei clienti** in mano, l'azienda può ora iniziare a rivolgersi ai clienti di ciascun gruppo separatamente con promozioni e offerte.
- Vale la pena notare che non sapevamo in anticipo quali caratteristiche avrebbe avuto ciascun gruppo, cioè non avevamo una risposta prima di fare effettivamente l'apprendimento non supervisionato. **Questa tecnica di apprendimento non-supervisionato è nota come raggruppamento (*clustering*) delle variabili**.

- Più tecnicamente, specialmente quando come visto a proposito delle RNC usiamo l'apprendimento non-supervisionato come tecnica di **pre-processamento dei dati** per l'inizializzazione dei pesi dei neuroni della rete nella fase di training del ML supervisionato invece dell'inizializzazione casuale, il **raggruppamento delle variabili in clusters** definisce in pratica il **numero dei gradi di libertà** – ovvero il numero delle dimensioni ortogonali dello spazio di probabilità – in cui può significativamente variare la **distribuzione di probabilità** che costituisce in ultima analisi la risposta del ML sia supervisionato che non-supervisionato.
- Quindi siccome lo scopo ultimo di ogni algoritmo di ML è la corrispondenza fra la distribuzione di probabilità in uscita dal sistema e la distribuzione di probabilità nascosta nei dati, **la corrispondenza (*matching*) dei gradi di libertà** fra quelli del sistema e quelli della base-dati è un aiuto notevolissimo che si dà al sistema **verso la convergenza ad un risultato ottimale** che minimizza la distanza fra a distribuzione in uscita dal sistema e la distribuzione nei dati. E/o, nel caso del ML supervisionato, **fra la distribuzione in uscita dal sistema e quella desiderata**.
- A parte il pre-processamento nei sistemi di ML supervisionato, le applicazioni di ML non-supervisionato sono quelle che riguardano:

- **La segmentazione dei clienti in classi** per applicazioni commerciali di profilamento dei clienti stessi.
- **La segmentazione delle immagini** per applicazioni intelligenti di computer grafica e/o di compressione intelligente delle immagini (p.es., per evitare la "squadrettatura" dell'immagine come nei primi algoritmi di compressione tipo *jpeg*.
- **L'analisi di mercato** per applicazioni finanziarie e/o commerciali intelligenti.
- **La scoperta (*detection*) di uso fraudolento dei dati (*fraud detection*)** in applicazioni intelligenti di sicurezza informatica.
- **La scoperta di anomalie** nell'uso, memorizzazione, distribuzione (*deployment*) dei dati in applicazioni intelligenti di sicurezza (*security*) e/o salvaguardia (*safety*) informatica…

## 6.2. ML quantistico (cfr. (Basti & Vitiello, 2022))

♦ È ovvio che esistano grandi aspettative verso l'uso di **computer quantistici nel ML** visto l'incremento esponenziale delle capacità di calcolo di un processore quantistico rispetto a quelli elettronici tradizionali.

- Tutto questo però è ancora al livello teorico e non ancora di applicazione pratica, visto le grandi difficoltà a realizzare computer quantistici che siano ***general purpose*** (effettivamente programmabili) e non solo ***special purpose.***
- Per esempio, grandi speranze e soprattutto timori (per la sicurezza informatica visto che tutti gli algoritmi di criptaggio si basano sulla non calcolabilità **in tempi polinomiali** (che non crescano cioè esponenzialmente ma solo linearmente) dell'algoritmo di fattorizzazione rispetto alla lunghezza della stringa numerica usata come chiave di criptaggio) aveva suscitato l'articolo pubblicato nel 1994 da Peter Shor (Shor, 1994) che riguardava la **fattorizzazione dei numeri interi in numeri primi.**
  - Shor ha dimostrato che su un computer quantistico questo algoritmo ha una **complessità computazionale solo polinomiale** (più esattamente **BQP (*Bounded Error Quantum Polinomial Time***).
  - Ovvero, i fattori sono trovati con un margine di errore arbitrariamente piccolo in tempo polinomiale (che cresce, cioè, linearmente e non esponenzialmente come nel caso classico)) rispetto alla lunghezza della stringa numerica da calcolare.

- Ebbene attualmente non esiste una macchina quantistica **scalabile** (ovvero la cui complessità può scalare in più o in meno rispetto alla complessità della funzione da calcolare) che implementi **in generale** l'algoritmo di Shor, mentre esistono versioni compilate (già scalate e/o ridotte per casi specifici) eseguite su **computer ottici quantistici** dell'algoritmo di Shor.
- Questo significa che le nostre reti di comunicazione sono criptate almeno finché non sia prodotto un computer quantistico capace di calcolare **in forma generale** l'algoritmo di Shor. Il pericolo comunque è incombente (si pensi alla sicurezza di dati bancari, di dati per il controllo delle centrali nucleari, del traffico aereo, del traffico cittadino, delle auto a guida autonoma, etc., etc.), almeno finché non dotiamo di un **criptaggio quantistico** le reti stesse…
- Per questo è essenziale sviluppare computer quantistici che **lavorino a temperatura ambiente** e che siano **facilmente usabili dall'utente finale istituzionale** per fini di criptaggio quantistico dei propri dati (darlo all'utente privato sarebbe pericoloso perché potrebbero usarlo anche i terroristi, rendendo impossibile intercettare le loro comunicazioni).

- In ogni caso, algoritmo di Shor a parte, alcune applicazioni di calcolo quantistico sono state sviluppate per risolvere il problema dei **minimi locali** in architetture di BP che usano il gradiente stocastico discendente.
  - Queste particolari architetture sono un'evoluzione quantistica delle cosiddette **Macchine di Boltzmann** che usano (pseudo) **fluttuazioni termiche** (rumore iniettato nell'algoritmo del gradiente discendente) per far sì che la dinamica "salti fuori" da un minimo locale. Una strategia che funziona se le "pareti" del minimo locale non sono troppo "alte", altrimenti il rumore da iniettare (fluttuazione termica) diventa troppo ed interferisce in maniera distruttiva con la ricerca stocastica (causale) del minimo globale.
  - L'evoluzione quantistica della Macchina di Boltzmann è la cosiddetta **Macchina di Boltzmann Quantistica** che invece di usare la fluttuazione termica per uscire da un minimo locale usa **l'effetto-tunnel quantistico** per sfuggire al minimo locale. Una strategia che funziona solo se le "pareti" del nimo locale non sono troppo "spesse", altrimenti l'effetto-tunnel non è applicabile…
- Recentemente col nostro gruppo abbiamo proposto un utilizzo di un computer quantistico basato non sulla QM ma sulla **QFT per sistemi dissipativi** come i nostri

cervelli (Vitiello, 1995; 2004) che usa il principio del **raddoppio dei gradi di libertà** fra sistema e ambiente (=base dati) come tecnica di pre-processamento non-supervisionato dei dati, particolarmente effettiva nel caso del **data-streaming** per la sua estrema velocità (Basti, Capolupo, & Vitiello, 2017; Basti & Vitiello, 2022)**.**

- Quando, cioè, si lavora su basi-dati **non fisse ma che evolvono nel tempo** (p.es., dati da rete internet, streaming dati di film; dati da reti di sensori ambientali, dati dei mercati finanziari, dati di sensori medicali, etc.), e dove dunque le dimensioni dello spazio di probabilità (= gradi di libertà) delle distribuzioni di probabilità nel **data-set dinamico** non sono fisse, ma **evolvono nel tempo**.
- **Il ML per il data-streaming** è infatti l'attuale limite o problema insolvibile per i classici algoritmi di ML supervisionati o no di tipo classico che non usano i modelli ML dei *Transformers* e che quindi in linea di principio lavorano molto bene per basi di dati anche molto grandi (*big-data*), ma comunque **fisse** (cfr. (Basti & Vitiello, 2022) per ulteriori spiegazioni).

# 7. Il modello di ML dei Transformers nell'IA generativa

## 7.1. I precursori dei Transformers nel NLP

- Come già detto al termine della Parte II, in meno di un decennio i Transformers hanno rivoluzionato il processamento innanzitutto di linguaggi naturali (*Natural Language Processing*) NLP, ma non solo, sbloccando capacità per i modelli di ML e quindi di IA che prima erano inimmaginabili.
- L'effetto più eclatante è stato lo sviluppo dirompente di **Chatbot (*chatting robot*** "o robot conversazionali"), ovvero di robot in grado di conversare intelligentemente con l'utente umano. Essi sono basati su un rivoluzionario modello di ML i cosiddetti ***Transformers*** ("trasformatori" ma è equivoco usare questa dizione) e, come ricordato fin da principio, hanno avuto un successo incredibile in soli tre anni (il primo modello di chatbot è stato ChatGPT-3, messo in linea su internet nel 2023), definendo una nuova classe di sistemi di IA quelli dell'**IA generativa** che vanno ben al di là

dell'applicazione al NLP e possono essere applicati a qualsiasi tipo di dati, non solo linguistici.

- L'architettura di rete neurale dei Transformers è infatti un'architettura di **calcolo parallelo a FFN** che ha rimpiazzato i modelli tradizionali di ML nel NLP e non solo, come **le reti neurali ricorrenti** (RNR) di **calcolo seriale o sequenziale,** dove l'informazione è propagata sia **in avanti** (*feedforward)* sia **all'indietro** (*feedbackward*) tipo la BP (cfr. Figura 12) o le RNC, con tutti i loro problemi legati al gradiente.
- Ma ha rimpiazzato anche **le reti neurali a memoria a breve e lungo termine** (LSTM) che hanno introdotto per la prima volta nelle RNR per il NLP un **meccanismo selettivo di attenzione**, mitigando senza però risolverli i problemi legati al gradiente.
- Eliminando l'uso di RNR (e quindi utilizzando solo reti neurali multistrato non-lineari a *feedforward* (FFN)) i Transformer usano **filtri molteplici ad alta parallelizzazione** e dunque fra loro indipendenti, migliorando notevolmente la capacità dei sistemi di IA di analizzare simultaneamente e secondo **molteplici contestualizzazioni** (sintattiche, semantiche e prammatiche) i dati linguistici, così da generare linguaggi

(non solo quelli naturali, ma anche formalizzati, nonché codici per la programmazione), altamente realistici e coerenti con i vari contesti d'uso.

- Invece, le architetture ML delle RNR sono progettate per elaborare dati **in sequenza** come,  e quindi intuitivamente hanno naturale applicazione, per esempio, per analizzare testo parlato o scritto, oppure serie temporali finanziarie nelle applicazioni dell'AI, in finanza, in cui **l'ordine degli elementi** è rilevante.
  - Infatti le RNR utilizzano connessioni ricorrenti, in cui l'output di un neurone in un passo viene reimmesso come input della rete nel passo successivo. Questo **calcolo sequenziale** consente alle RNR di catturare le dipendenze temporali e i modelli all'interno delle sequenze.
  - Tuttavia, le RNR tradizionali soffrono, come sappiamo, del problema del gradiente evanescente (cfr. § 5.2.4) che nel caso del NLP diviene particolarmente rilevante nell'elaborazione di lunghe sequenze di testo (i gradienti a lungo termine che vengono propagati all'indietro possono tipicamente "scomparire", cioè, tendere a zero, interrompendo così l'apprendimento). Ciò limita la loro capacità di apprendere **dipendenze a lungo raggio**.

- Questo limite è stato affrontato in passato con lo sviluppo delle **architetture LSTM** introdotte nel 1997, che sono diventate le architetture RNR standard per l'elaborazione delle dipendenze a lungo termine (Sherstinsky, 2020). La relativa insensibilità alla lunghezza dell'intervallo è il vantaggio delle LSTM rispetto ai modelli RNR classici e ad altri modelli ML sequenziali.
- I modelli LSTM mirano infatti **a fornire una memoria a lungo termine (Long-*Term Memory,* LTM)** per RNR che può durare migliaia di passi, da cui il nome di "memorie a lungo termine" per questa architettura RNA. Un nome scelto in analogia con la distinzione tra memoria a lungo termine e memoria a breve termine utilizzata nelle neuroscienze cognitive.
- L'intuizione alla base dell'architettura LSTM è quella di **creare un modulo aggiuntivo a quello LTM nella rete neurale,** che impari quando ricordare e quando dimenticare le informazioni che potrebbero **non essere più necessarie** per ulteriori calcoli. Ovvero, una **memoria a breve termine (*Short-Term-Memory,* STM**).
    - **Ad esempio**, un'architettura LSTM può apprendere correttamente le dipendenze grammaticali nello NLP. Quando elabora la frase "Giovanni, a causa del suo comportamento controverso, è ora un paria nel gruppo", la rete, ricordando il

genere grammaticale (**maschile**/femminile) e il numero (**singolare**/plurale) del soggetto “Giovanni”, è in grado **di conservare questa informazione** come pertinente per il pronome “suo”, **ma non più rilevante** dopo il verbo in terza persona singolare “è”.

- Questo tipo di **meccanismo sequenziale di attenzione** per selezionare statisticamente le informazioni rilevanti da memorizzare è implementato, dotando ogni unità LSTM (cella) di **tre gate**: un **gate di input**, un **gate di output** e un **gate di dimenticanza** che regolano il flusso di informazioni in entrata e in uscita dalla cella.
  - **I gate di dimenticanza** decidono quali informazioni scartare dallo stato precedente mappando l'input corrente su un valore compreso tra 0 (scarto) e 1 (conservazione).
  - **I gate di input** decidono quali nuove informazioni memorizzare nello stato della cella, utilizzando lo stesso tipo di mappatura dei gate di dimenticanza.
  - **I gate di output** decidono quali informazioni dello stato attuale della cella emettere, assegnando un valore compreso tra 0 e 1 alle informazioni, considerando gli stati precedenti e attuali.

- In questo modo, emettendo in modo selettivo le informazioni rilevanti dallo stato attuale, la rete neurale LSTM è in grado di **mantenere utili dipendenze a lungo termine** per effettuare previsioni, sia nel tempo attuale che in quello futuro sulla nuova parola da produrre.
- **Un ulteriore miglioramento** dell'efficienza computazionale del modello LSTM è costituito dall'introduzione nel 2014 della cosiddetta **unità ricorrente *gated* (*Gated Recurrent Unit* GRU)** nelle RNR, che è un meccanismo di gating più efficiente rispetto a quello utilizzato nelle LSTM standard.
  - La GRU è infatti simile a un'unità LSTM con un meccanismo di gating per inserire o dimenticare alcune caratteristiche, **ma manca del vettore di contesto** rappresentato dal gate di output nelle unità LSTM.
  - In questo modo, la GRU deve elaborare un numero inferiore di parametri rispetto a un'unità LSTM, in modo da mitigare il problema **dell'esplosione del gradiente** che affligge i classici LSTM in quanto sono comunque delle RNR (cfr. § 5.2.4).

## 7.2. La rivoluzione concettuale dei Transformers

- Sebbene M. Minsky non abbia bisogno del nostro riconoscimento per sottolineare il suo genio che lo rende uno dei precursori dell'IA, la rivoluzione concettuale dei Transformers nei modelli ML ha una parola chiave per comprenderne la potenza computazionale e persino cognitiva: **parallelizzazione**.
- Ciò dimostra che Minsky aveva ragione quando nella Seconda Edizione del suo *Perceptrons* (Minsky & Papert, 1987) criticava l'approccio PDP, e in particolare il modello ML della BP, perché **non sono architetture di calcolo adeguatamente parallele**, a differenza del perceptrone di Rosenblatt e della sua prima versione multistrato di A. Gamba.
- In effetti, come vedremo, la presenza di diverse architetture FFN del perceptrone multistrato **è una caratteristica chiave delle architetture dei Transformers**, anche se i loro neuroni sono arricchiti **con funzioni di attivazione non lineari ReLu** (cfr. § 5.3.2) per catturare correlazioni di ordine superiore nel loro input.
- Tuttavia, il nucleo parallelo delle architetture dei Transformers, da cui dipende la loro efficacia, è correlato al loro **meccanismo di auto-attenzione**, giustificando lo slogan

"l'attenzione è tutto ciò di cui hai bisogno" nell'articolo citato che li ha imposti all'attenzione generale (Vaswani, et al., 2017).

- In breve,
  - Mentre il meccanismo di attenzione negli LSTM è tipicamente un meccanismo supplementare che aiuta **sequenzialmente** un'architettura RNR codificatore (per la rappresentazione/analisi dell'input) - decodificatore (per la generazione dell'output) **a concentrarsi in modo sequenziale su diverse parti dell'input per un dato output**,
  - **L'auto-attenzione** è una componente fondamentale dell'architettura dei Transformers che consente loro **di prestare attenzione simultaneamente** a diversi contesti interpretativi per la stessa sequenza di input **per catturare simultaneamente, piuttosto che in modo sequenziale, le dipendenze nell'intero input**, al limite dell'intero vocabolario di un LLM con centinaia di migliaia di vocaboli.
  - Per questo motivo in passato è stato definito anche come **meccanismo di attenzione di ordine superiore**, anche se tale connotazione è logicamente fuorviante. In breve, l'auto-attenzione consente ai Transformers di elaborare gli input in modo **olistico**, ottenendo una migliore comprensione contestuale e consentendo

il calcolo parallelo, a differenza della natura intrinsecamente sequenziale degli LSTM.

- In questo modo, l'architettura del Transformer ha cambiato radicalmente i modelli linguistici di grandi dimensioni (LLM) e non solo, fornendo una struttura versatile adattabile a un'ampia gamma di attività di elaborazione del linguaggio naturale (NLP). Ma anche, in seguito, **di immagini, video, audio e azioni** nel caso di robot.
- A seconda dell'attività da svolgere, che si tratti **di comprendere, generare o tradurre un testo**, gli LLM basati sui Transformers utilizzano una delle tre principali configurazioni del Transformer:

1. **Modelli solo codificatore**,
2. **Modelli solo decodificatore** e
3. **Modelli codificatore-decodificatore**.

- Ciascuna configurazione presenta punti di forza distinti ed è ottimizzata per applicazioni specifiche.

1. **Modelli solo codificatore** (ad esempio, **il modello BERT** di Google, che sta per ***Bidirectional Encoder Representation for Transformers***. Sono progettati e

ottimizzati per **compiti di comprensione e rappresentazione** senza la necessità di generare nuovi testi. I modelli solo encoder sono ottimi per comprendere le connessioni complesse nel testo. Lo fanno creando incorporamenti di dati linguistici di alta qualità. Poiché elaborano il testo in entrambe le direzioni, questi modelli sono in grado di cogliere **il contesto completo** di una parola osservando ciò che la precede e la segue in una frase e/o in lunghe sequenze di frasi.

- Sono ottimizzati per attività che richiedono l'analisi e la classificazione di testi, quali:
  a. **Classificazione delle frasi**: determinare l'argomento di una frase, ad esempio il suo tono generale, il soggetto o lo scopo.
  b. **Analisi del sentiment:** capire se un testo è positivo, negativo o neutro.
  c. **Riconoscimento di entità denominate**: individuare nomi di persone, aziende e luoghi in un testo.
  d. **Risposta alle domande**: fornire risposte alle domande sulla base di un determinato testo.
- **Applicazioni nel mondo reale**: un ottimo esempio è BERT applicato al **motore di ricerca di Google**. Ha migliorato in maniera significativa i risultati di ricerca

comprendendo il contesto di ciò che si sta cercando, il che aiuta a trovare informazioni più accurate e pertinenti.

2. **Modelli solo decoder**, come la serie **GPT**. Sono progettati per generare testo.
   - A differenza dei modelli che elaborano le informazioni in entrambe le direzioni, i modelli solo decodificatori **funzionano in una sola direzione**, da sinistra a destra. Questo li rende ottimali per compiti che implicano **la previsione di ciò che viene dopo in una sequenza**, garantendo che il testo generato sia coerente e abbia senso in una conversazione con l'utente.
   - **Punti di forza e applicazioni.** Questi modelli sono particolarmente efficaci nel creare scambi linguistici naturali e fluidi a partire da un dato prompt. La loro capacità di prevedere la parola successiva li rende ideali per varie applicazioni:
     - **AI conversazionali**. Utilizzata nelle chatbots e negli assistenti virtuali tipo Alexa per creare dialoghi simili a quelli umani.
     - **Scrittura creativa**. Possono aiutare a scrivere storie, poesie, sceneggiature cinematografiche e televisive e altri contenuti creativi.

        - **Generazione di contenuti**. Possono produrre articoli, post sui social media o riassunti a partire da un breve prompt.
        - **Generazione di codice**. Sono persino in grado di trasformare descrizioni in linguaggio naturale in **codice informatico** (programma di calcolatore nei vari linguaggi di programmazione).
    - **Applicazioni nel mondo reale**. La serie Chat-GPT (divenuta multimodale perché si applica non solo ai testi ma anche ad immagini, video, audio e alla loro combinazione in Chat-GPT-4/5) è un ottimo esempio di questo tipo di modelli che, come spiegato in precedenza, hanno inaugurato una nuova era **nell'IA di largo consumo** determinando il cosiddetto **boom dell'IA** cui stiamo assistendo.
3. **I modelli encoder-decoder**, come **il modello T5** (*Text-To-Text Transfer Transformer*), presentati nel 2020 da un gruppo di ricercatori di Google in un articolo in cui sottolineano che "il nostro obiettivo non è quello di proporre nuovi metodi, ma piuttosto di fornire una prospettiva completa sullo stato attuale del settore" [? ].

- In effetti, i modelli encoder-decoder integrano i due tipi precedenti di Transformers e sono quindi **incredibilmente versatili** perché combinano sia componenti encoder che decoder.
- Questo design è ottimale per attività che richiedono **la trasformazione di una sequenza di testo in una completamente diversa**, come **la traduzione, la sintesi o la parafrasi**.
- L'encoder elabora prima il testo di input **per comprenderne il significato**; quindi, il decoder utilizza tale comprensione **per creare l'output finale**.
- **Punti di forza e applicazioni**. Questi modelli sono efficaci perché in grado di gestire una varietà di compiti che richiedono sia la comprensione del contesto dell'input che la creazione di un output coerente. Ciò li rende ideali per compiti complessi che richiedono una conversione specifica da input a output. Tra le applicazioni più comuni figurano:
    - **Traduzione automatica**, che era una vera e propria bestia nera agli albori dell'IA e che grazie alla tecnologia encoder-decoder dei

Transformers ha raggiunto oggi un'incredibile precisione e versatilità che **ha reso obsoleto il lavoro dei traduttori umani**.

- **Sintesi di testi**: condensare articoli lunghi in sintesi più brevi.
- **Trasformazioni di testo:** riformulazione di frasi, trasformazione di affermazioni in domande o completamento di testi mancanti.

- **Applicazioni nel mondo reale**. Un esempio ben noto è **il modello fondamentale T5.** È famoso per la sua capacità di gestire contemporaneamente molteplici compiti diversi, trattando ogni problema come una semplice conversione da testo a testo. Ciò rende T5 uno strumento molto flessibile (modello-base) per tutti i tipi di compiti linguistici.
- Infine, **l'estensione dei modelli Transformer** dalle attività di NLP all'elaborazione di immagini e video, nonché all'elaborazione di audio e musica grazie alla loro integrazione con i modelli RNC, ha inaugurato l'era **dei modelli di IA generativa multimodale**. La motivazione principale per combinare RNC e Transformer è quella di sfruttare i punti di forza di ciascuna architettura mitigandone al contempo i punti deboli.

- Le RNC eccellono **nel catturare le caratteristiche locali e le gerarchie spaziali** grazie alle loro operazioni convolutive e **alle capacità induttive** (come la località e l'equivarianza per traslazione).
- Al contrario, i Transformers sono potenti nel modellare **il contesto globale e le dipendenze a lungo raggio** su **un'intera immagine**, su **video e/o sequenze audio o musicali** utilizzando il loro meccanismo di auto-attenzione.
- Questi modelli ibridi utilizzano in genere una RNC come **estrattore di caratteristiche** per ottenere una **rappresentazione locale "densa di particolari (*features*)"**; quindi, inseriscono queste caratteristiche in un Transformer per catturare le relazioni globali.
- **Alcune applicazioni significative** già sviluppate di queste architetture ibride nell'imaging e nell'elaborazione video riguardano, ad esempio, **l'imaging medico** per la diagnosi dei tumori, le applicazioni per **il riconoscimento delle azioni umane dai video** delle telecamere di sorveglianza.

- **Le applicazioni nell'elaborazione audio e musicali** riguardano il riconoscimento e la generazione automatica del parlato e l'elaborazione e la generazione automatica della musica.
- Ma, naturalmente, **siamo solo all'inizio** dello sviluppo delle applicazioni generative di IA multimodale, i cui confini sono oggi impossibili da definire e persino da immaginare.

## 7.3. Una panoramica dell'architettura dei modelli di tansformers negli LLM

♦ Partiamo dalla seguente Figura 16 che mostra **l'architettura di base di un modello di *Transformer* per LLM** con *stack* di codifica e decodifica, come riportato nell'articolo citato *Attention is all you need* all'origine della rivoluzione dei tansformers (Vaswani, et al., 2017). Nel resto di questa sottosezione illustreremo i diversi componenti dell'architettura mostrata in figura.

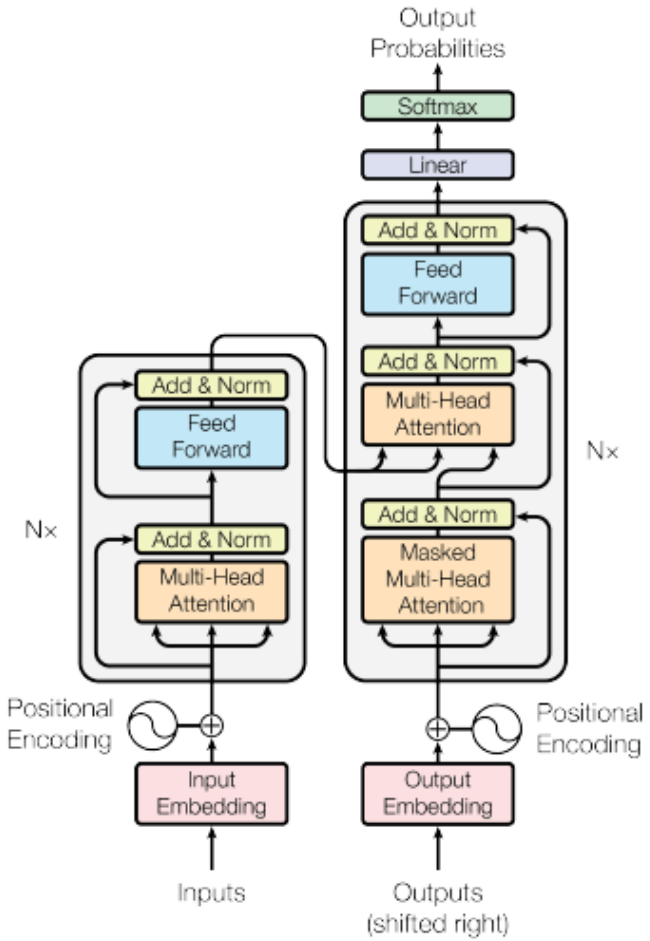



*Figura 16. Il Transformer: architettura-base del modello (Vaswani, et al., 2017)*

### 7.3.1. Tokenizzazione e incorporamento

- **Tokenizzazione**. Innanzitutto, il testo viene **suddiviso in parti più piccole** chiamate ***token*** **codificati numericamente**, che possono essere parole intere, parti di parole o singoli caratteri. Questo processo, noto come tokenizzazione, aiuta il modello a elaborare in modo efficiente diversi tipi di testo e lingue, **catturando anche i minimi dettagli**. Nel caso **dell'elaborazione di immagini o video**, sono le sequenze di immagini o video ad essere suddivise in parti più piccole chiamate ***patch***. Il principio è comunque lo stesso.
- **Incorporamento (*embedding*).** Ogni token viene convertito in un **vettore numerico multidimensionale**. Questi incorporamenti contengono informazioni sul significato del token e sulla sua relazione con altri token. Ad esempio, gli incorporamenti per "maschio" e "femmina" sarebbero più vicini tra loro nello spazio vettoriale rispetto agli incorporamenti per "maschio" e "camion", mostrando la loro somiglianza semantica. Ciò fornisce al modello una **rappresentazione numerica utile** per ogni token.

### 7.3.2. Codifica posizionale.

- Poiché i Transformers elaborano tutti i token in parallelo, non hanno una comprensione intrinseca **dell'ordine delle parole e/o dei token**, come nell'elaborazione

sequenziale delle RNR. Per risolvere questo problema, a ciascun token viene aggiunto **un vettore di codifica posizionale unico**. Questo vettore viene generalmente creato utilizzando **le funzioni seno e coseno**, come mostrato dal simbolo utilizzato nella Figura 15, e questo aiuta il modello a riconoscere la posizione di ciascun token nella sequenza.

- **Esempio:** la codifica posizionale aiuta il modello a comprendere la differenza tra frasi come: "l'auto segue il camion" e "il camion segue l'auto".

### 7.3.3. Calcolo dell'auto-attenzione.

- Come sappiamo, alla base del suo funzionamento, il Transformer utilizza un meccanismo di auto-attenzione **per capire quanto sia importante ogni token rispetto agli altri, secondo diverse "finestre" di attenzione (contesti)** in una sequenza. Ciò consente al modello di comprendere le connessioni e le dipendenze tra i token, indipendentemente dalla loro distanza nello spazio vettoriale. In altri termini, il calcolo dell'auto-attenzione consente al modello di valutare ogni token in relazione a tutti gli altri token in una sequenza, consentendo al modello di **comprendere il contesto e le relazioni** indipendentemente dalla posizione nella sequenza.

- **Attenzione mediante prodotto scalare**. Per illustrare il meccanismo di "auto-attenzione" utilizzato nei Transformers, gli autori di (Vaswani, et al., 2017) hanno associato a ciascun token **tre vettori particolari**:
  - **I vettori di query Q** che rappresentano le "richieste" di informazioni del token;
  - **I vettori di chiave (*key*) K** che rappresentano ciò che ciascun token "offre" in risposta e
  - **I vettori di valore V** che rappresentano le informazioni effettive trasmesse dal token.
  - Tutti questi vettori per il calcolo della funzione di attenzione sono "raggruppati" **nelle rispettive matrici Q, K, V**. In questo modo, la funzione di attenzione calcolata che si trova nei Transformers (cfr. (Vaswani, et al., 2017) p. 4) viene così descritta dagli Autori:

    «L'input è costituito da query e chiavi di dimensione $d_k$ e valori di dimensione $d_V$ . Calcoliamo i prodotti scalari della query con tutte le chiavi, dividiamo ciascuno per $d_k$ e applichiamo la funzione *softmax* per ottenere i pesi sui valori. In pratica, calcoliamo la funzione di attenzione su un insieme di

query simultaneamente, raggruppate in una matrice **Q**. Anche le chiavi e i valori sono raggruppati in matrici **K** e **V**. Calcoliamo perciò la matrice degli output come:

$$\textbf{Attenzione}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \ softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Dove il fattore di scala $\frac{1}{\sqrt{d_k}}$ è necessario per evitare che per valori elevati di $d_k$ i prodotti scalari crescano troppo così da confinare la funzione softmax in regioni in cui presenta gradienti estremamente ridotti».

- **Funzione Softmax**. Infatti, la funzione *softmax* è generalmente una funzione che prende come input una *n*-pla *z* di *K* numeri reali e la normalizza in una distribuzione di probabilità costituita da *K* probabilità proporzionali agli **esponenziali** dei numeri di input. In questo modo, il termine "softmax" deriva **dagli effetti amplificatori dell'esponenziale** su qualsiasi massimo nella *n*-pla di input, quindi dovrebbe essere correttamente indicata come *softargmax* perché non è un "massimo regolare", ovvero un'approssimazione regolare della funzione massima.

- Invece, la funzione softmax prende un insieme di numeri reali (=argomento) e li trasforma in una distribuzione di probabilità. Lo fa applicando prima la funzione esponenziale a ciascun numero. Quindi, normalizza questi nuovi valori dividendo ciascuno di essi per la somma di tutti gli esponenziali. Questo processo garantisce che i numeri nell'output finale sommati diano come risultato **1**'**enfatizzazione di alcuni valori,** quelli "attenzionati" dal modello. Il nome "softmax" (correttamente "softargmax") deriva dal modo in cui **la funzione esponenziale enfatizza il numero più grande nell'input** quindi nel suo argomento, facendolo risaltare ancora di più nell'output finale. Ad esempio, nella *n*-upla (1, 2, 8), l'output softmax è approssimativamente (0,002, 0,006, 0,992), che assegna quasi tutto il peso al numero più grande, 8. Questo spiega **il suo ruolo nel meccanismo di auto-attenzione del Transformer**.
- Per capire il senso dell'esponente *T* nella formula, basta ricordare – senza qui entrare in particolari – che un famoso esempio di funzione *softargmax* in meccanica statistica è la **distribuzione di Boltzmann** che fornisce una **definizione termodinamica** della temperatura *T*, come $\beta =$

$1/kT$ e quindi come l'inverso della temperatura $T$, dove $k$ è la costante di Boltzman. Ovviamente, un valore $\beta$ più alto (temperatura più bassa) rende la distribuzione dell'output più casuale e uniforme, cioè con **un'entropia più alta**, mentre un valore $\beta$ più basso (temperatura più alta) rende l'output più nitido e meno casuale, con **un valore che diventa più prominente**.

- Questa analogia termodinamica, da un lato, spiega il significato dell'esponente $T$ nella formula dell'attenzione dei Transformer, dall'altro, spiega perché nei modelli di Transformer la **mappa dei pesi statistici dei diversi token,** dopo l'applicazione del meccanismo di auto-attenzione, è definita come **una mappa di calore (*heat-map*)**, che svolge un ruolo essenziale nella generazione dell'output dello stack del decodificatore di un modello di Transformer. Intuitivamente, il token di output **è il più "caldo"** nella mappa termica dei tokens.

### 7.3.4. Attenzione multi-head e reti feed-forward.

♦ La parallelizzazione dei calcoli di auto-attenzione nei Transformers ha forse la conseguenza più importante nel fatto che la sequenza di input **può essere contestualizzata**

**simultaneamente secondo molteplici "finestre di attenzione"** (*multihead attention*)

- **Attenzione multi-head.** Per comprendere i vari modelli linguistici, il meccanismo di auto-attenzione viene replicato **su diverse finestre (teste) di attenzione**. Ogni **"testa" impara a concentrarsi su diversi aspetti del testo**. Ad esempio, una testa potrebbe analizzare la struttura sintattica del testo, mentre un'altra potrebbe concentrarsi sul significato complessivo o su relazioni semantiche più sottili. Citando di nuovo (Vaswani, et al., 2017):
  «L'attenzione multi-head consente al modello di prestare attenzione congiuntamente alle informazioni provenienti da diversi sottospazi di rappresentazione in posizioni diverse. Con un singolo ***head* di attenzione**, invece, l'operazione di media inibirebbe questo processo». La concatenazione di diverse "teste" di attenzione consente così al modello di produrre **un'unica matrice di output dei pesi $W^O$** che tiene conto simultaneamente della rilevanza dei token per molteplici meccanismi di attenzione.
- **Reti a *feed-forward* (FNN).** Dopo essere passati attraverso il meccanismo di auto-attenzione multi-head, **i token vengono inviati ad una FFN multi-strato di neuroni con funzioni non-lineari di attivazione** (generalmente delle funzioni ReLu). Questo

consente al modello di migliorare la complessità e la raffinatezza delle rappresentazioni/interpretazioni dei token individuando fra di loro **correlazioni complesse**. Uno strato, a un dato livello, elabora ogni token individualmente per apprendere modelli linguistici complessi e, quindi, tutti gli output di uno strato convergono nello strato (o negli strati) del livello successivo.

### 7.3.5. Stratificazione dei livelli.

- I Transformers sono costituiti così da **diversi strati (*layers*) sovrapposti.** Ogni strato dispone sia di un meccanismo di attenzione multi-head che di una rete neurale feed-forward. Man mano che i dati di input attraversano questi strati, la comprensione del testo da parte del modello diventa sempre più complessa. Ogni strato aiuta il modello a comprendere meglio **le relazioni tra i token**, consentendogli di cogliere **modelli linguistici dettagliati e sottili**.

### 7.3.6. Meccanismi di auto-attenzione "mascherata" nelle architetture solo-decoder.

- Per quanto riguarda l'architettura del Transformer, **dobbiamo distinguere tra diversi meccanismi di auto-attenzione**, a seconda dell'architettura complessiva, rispettivamente per le architetture **encoder-decoder, solo-encoder e solo-decoder**.

- **Negli strati di "attenzione encoder-decoder"**. Le query provengono dal livello decoder precedente e le chiavi e i valori della memoria provengono dall'output dell'encoder. Ciò consente a ogni posizione nel decoder di **prestare attenzione a tutte le posizioni nella sequenza di input**. In questo senso, si parla di **un meccanismo di auto-attenzione bidirezionale**.
- **Negli strati di auto-attenzione "solo codificatore".** In questa architettura, in uno strato di auto-attenzione, le chiavi, i valori e le query provengono tutti dalla stessa fonte, **in particolare dall'output del livello codificatore precedente**. Ciò significa che ogni posizione nel codificatore può prestare attenzione a tutte le posizioni, in entrambe le direzioni nel livello precedente.
- **Gli strati di auto-attenzione "solo decodificatore".** Negli strati di auto-attenzione del decodificatore, ogni posizione può concentrarsi **solo sulle posizioni che la precedono nella sequenza**, ma non su quelle che la seguono. Questo serve a mantenere **la proprietà auto-regressiva** del modello, che permette di prevedere la parola successiva sulla base delle parole già viste.
    - Per ottenere questo risultato, il modello di auto-attenzione utilizza una tecnica chiamata **mascheramento**.

«Implementiamo questo all'interno dell'attenzione del prodotto scalare **mascherando** tutti i valori nell'input del softmax che corrispondono a connessioni illegali, ovvero **le parole "dopo" la posizione effettiva**. Questo impedisce efficacemente al modello di considerare informazioni future, assicurando che il modello guardi solo ciò che avrebbe dovuto vedere fino a quel momento, **nel prevedere la parola successiva**. In questo senso, dobbiamo parlare di **un meccanismo di auto-attenzione unidirezionale** nelle architetture dei **Transformers-solo-decodificatori**. Questo meccanismo di auto-attenzione mascherato è il cuore dei modelli ML generativi basati sui Transformers, da cui dipende il loro successo».

### 7.3.7. FFN basate sulla posizione.

- Infine, tutto ciò significa che il modello di auto-attenzione deve essere consapevole della posizione relativa dei token che sta elaborando. Dato che, dopo l'applicazione multipla della funzione softmax, le informazioni della "codifica posizionale" iniziale sono andate perse. Queste informazioni di posizione relativa di "ordine superiore" nel meccanismo di auto-attenzione si ottengono perché, oltre ai sottostrati di attenzione, ogni strato dello stack nel codificatore e nel decodificatore **contiene anche una rete feed-forward completamente connessa**. Questa rete viene applicata a ciascuna

posizione in modo indipendente e consiste **in due trasformazioni lineari con una funzione di attivazione ReLU** tra di esse:

$$FFN(x) = max(0, xW_1 + b)W_2 + b_2$$

- In questo modo, mentre le trasformazioni lineari sono le stesse in posizioni diverse, **utilizzano parametri diversi da uno strato all'altro**. Ciò fornisce al modello, in modo molto elegante, le informazioni posizionali di "ordine superiore" di cui ha bisogno.
- **Output dei modelli generativi.** Nei modelli LMM incentrati su compiti generativi (ad esempio la serie GPT), il modello genera il linguaggio **prevedendo il token successivo sulla base dell'input elaborato**. Il modello utilizza uno strato softmax per calcolare le probabilità **su un LLM pre-addestrato sull'intero vocabolario**, perfezionandolo utilizzando il proprio **meccanismo di auto-attenzione multi-head mascherato multistrato (solo decodificatore)**, selezionando il token con la probabilità più alta come parola successiva. Questo processo viene ripetuto iterativamente per generare un testo coerente e contestualmente appropriato.

### 7.3.8. Modelli di base nei Transformers.

- Oltre al GPT per i modelli di Transformers solo decodificatori, esiste una varietà sempre più ampia di **architetture di Transformers-base pre-addestrate** che fungono da modelli di base per uso generico, **successivamente ottimizzati o adattati** per un'ampia gamma di attività a valle.
- In altre parole, quando si sviluppa un nuovo modello di Transformer per un'applicazione specifica, **non si deve partire da zero**, ma da **un modello di base pre-addestrato** su una quantità molto grande di dati che fornisce **una serie iniziale di parametri adeguati da cui partire**. Questi modelli di base esistono per tutte le principali classi di Transformers illustrate finora.

1. **Transformers solo encoder (serie BERT)**. Oltre al modello BERT bidirezionale originale, esistono **RoBERTa (*Robustly Optimized BERT Pretraining Approach*)** che si basa su BERT rimuovendo il compito di previsione della frase successiva, addestrando più a lungo, su più dati e utilizzando il mascheramento dinamico.
    a. **ALBERT (*A Lite BERT*)** che riduce le dimensioni del modello tramite la condivisione dei parametri e gli embedding fattorizzati, mantenendo le prestazioni con un numero inferiore di parametri.

b. **ELECTRA (*Efficiently Learning an Encoder that Classifies Token Replacements Accurately*)** che addestra l'encoder come discriminatore per rilevare i token sostituiti, ottenendo rappresentazioni più forti con meno calcoli.

c. Esistono anche **varianti specifiche per dominio**, come **BioBERT / SciBERT** per testi biomedici e scientifici; **CodeBERT / GraphCodeBERT** per la comprensione del codice sorgente; **LayoutLM** per la comprensione di moduli e documenti e così via.

2. **Transformers Encoder-Decoder.** Oltre al modello T5 originale, i principali modelli-base ampiamente utilizzati in questa classe di Transformers sono:

   a. **BART (*Bidirectional and Auto-Regressive Transformers*)** che corrompe il testo con rumore (cancellazioni, rotazioni) e addestra un **modello seq2seq** per ricostruire l'originale, efficace per la sintesi e la generazione di testo;

   b. **mT5 (*Multilingual T5*)** che estende T5 a oltre 100 lingue, apprendendo un modello unificato da testo a testo che estende i modelli di traduzione di trasferimento tra le lingue.

3. **Modelli di base in altre modalità.** Oltre al testo puro, il paradigma del pre-addestramento **si estende ai modelli visivi, vocali e cross-modali.** Anche questi costituiscono la base per perfezionamenti specializzati. Alcuni esempi sono:
    a. **ViT (*Vision Transformer*)** che applica un codificatore Transformer puro a patch di immagini, pre-addestrato su enormi set di dati di immagini, quindi ottimizzato per la classificazione, il rilevamento e la segmentazione.
    b. **wav2vec e HuBERT** che apprendono le rappresentazioni vocali prevedendo segmenti audio mascherati. Possono essere ottimizzati per il riconoscimento vocale automatico, l'identificazione del parlante e il riconoscimento delle emozioni.
    c. **CLIP e ALIGN** che addestrano congiuntamente codificatori di immagini e testo per allineare i loro embedding, consentendo la classificazione delle immagini zero-shot e il recupero cross-modale.

# Bibliografia della II Parte


Aczel, P. (1988). *Non-well-founded sets. (CLSI Lecture Notes, vol. 14).* Stanford CA: Stanford UP.

Basti, G., & Vitiello, G. (2022). A QFT Approach to Data Streaming in Natural and Artificial Neural Networks. *Proceedings, 81*, 106. doi:10.3390/proceedings2022081106

Basti, G., Capolupo, A., & Vitiello, G. (2017). Quantum Field Theory and Coalgebraic Logic in Theoretical Computer Science. *Prog. in Bioph. & Mol. Biol. Special Issue: Quantum information models in biology, from molecular biology to cognition, 130*, 39-52.

Ghosh, A., Sufian, A., Sultana, F., Chakrabarti, A., & Debashis, D. (2020). Fundamental Concepts of Convolutional Neural Network. In V. E. Balas, R. Kumar, & R. Srivastava (Eds.), *Recent Trends and Advances in Artificial Intelligence and Internet of Things. Intelligent Systems Reference Library, vol 172* (pp. 519-567). Berlin, New York, Germany, USA: Springer. doi:10.1007/978-3-030-32644-9_36

Kozma, R., Puljic, M., & Freeman, W. J. (2014). Thermodynamic Model of Criticality in the Cortex Based on EEG/ECoG Data. In D. Plenz, & E. Niebur (Eds.), *Criticality in Neural Systems* (pp. 153-176). Hoboken NJ, USA: Wiley-VCH Verlag GmbH & Co. KGaA.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *NIPS'12: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1. December 2012* (Vol. 1, pp. 1097-1105). ACM. doi:10.1145/3065386

McCulloch, W. S., & Pitts, W. H. (1943). A Logical Calculus of Ideas Immanent in Nervous Activity. *Bull. Math. Biophys., 5*, 115-133.

Minsky, M. (1975a). A framewok for representing knowledge. In P. H. Winston (Ed.), *The psychology of the computer vision.* New York: McGraw-Hill.

Minsky, M. (1975b). Frame system theory. *TINLAP.* Boston MA, USA: MIT Press.

Minsky, M. (1977). Frame theory. In P. Johnson-Laird, & P. Wason (Eds.), *Thinking: Reasings in Cognitive Science* (pp. 355-376). Cambridge, MA: Cambridge UP.

Minsky, M. (1986). *The Society of Minds.* New York, USA: Simon & Schuster.

Minsky, M., & Papert, S. (1987). *Perceptrons. An introduction to computational geometry. Second Edition.* Cambridge MA, USA: MIT Press.
Nwankpa, C. E., Ijomah, W., Gachagan, A., & Marshall, S. (2018, November 8). *Activation functions: comparison of trends in practice and research for deep learning.* doi:10.48550/arXiv.1811.03378
Putnam, H. (1960). Minds and Machines. In S. Hook (ed.), *Dimensions of mind.* New York: Collier.
Pylyshyn, Z. W. (1986). *Computation and Cognition: Toward a Foundation for Cognitive Science.* Cambridge MA, USA: MIT Press.
Quine, W. V. (1987). *Quiddities. An Intermittently Philosophical Dictionary.* Cambridge MA, USA: Harvard UP.
Rosenblatt, F. (1961). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Buffalo, NY: Cornell UP.
Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors, Nature, 1986, 323, pp. 533-536. *Nature, 323*, 533-536.
Rumelhart, D., McClelland, L. J., & PDP Group. (1986). *Parallel Distributed Processing. Voll. 1-2.* Cambridge MA, USA: MIT Press.

Searle, J. R. (1980). Minds, brains, and programs. *The Behav. and Brain Sc., 3*, 128-135.

Searle, J. R. (1983). *Intentionality. An essay in the philosophy of mind.* New York: Cambridge UP.

Sherstinsky, A. (2020). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena, 404*(132306).

Shor, P. (1994). Algorithms for quantum computation: discrete and log factoring. *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science* (pp. 124-134). Santa Fe CA, USA: IEE Computer Society Press.

Turing, A. M. (1937). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, 2*(42), 230–265.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind, 59*, 433-460.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). Attention is all you need. *Adv. in Neur. Inf. Proc.*(30).

Vitiello, G. (1995). Dissipation and memory capacity in the quantum brain model. *Int. J. Mod. Phys., B9*, 973-989.

Vitiello, G. (2004). The dissipative brain. In G. G. Globus, K. H. Pribram, & G. Vitiello (Eds.), *Brain and Being - At the boundary between science,philosophy,language and arts* (pp. 317-330). Amstedam: John Benjamins Pub. Co.

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. PhD thesis.* Boston MA, USA: Harvard UP.

# Note

[1] Per una presentazione elementare ma precisa di tutto quanto diremmo in questo capitolo ci si può ultimamente riferire al sito MLK: https://machinelearningknowledge.ai ed in particolare alla sezione in cui tratta i «primitivi dei ML» e cioè i modelli del neurone artificiale di McCulloch & Pitts: Neural Network Primitives Part 1 - McCulloch Pitts Neuron Model (1943) - MLK - Machine Learning Knowledge; del perceptrone Neural Network Primitives Part 2 - Perceptron Model (1957) - MLK - Machine Learning Knowledge; del neurone artificiale con funzione di attivazione sigmoidale: Neural Network Primitives Part 3 – Sigmoid Neuron - MLK - Machine Learning Knowledge; degli attuali modelli di neurone artificiale le cui funzioni di attivazione vanno oltre il neurone con modello di attivazione sigmoidale e basato sull'algoritmo di apprendimento del «gradiente stocastico» per le sue intrinseche limitazioni: Neural Network Primitives Final Part 4 - Modern Artificial Neuron - MLK - Machine Learning Knowledge