Autore: Gabriel Rovesti

Prof. Marco Zorzi, Prof. Alberto Testolin
Università degli Studi di Padova

INDICE GENERALE

---

1. INTRODUZIONE ALL'INTELLIGENZA ARTIFICIALE

1.1 Definizione e Storia dell'Intelligenza Artificiale

Definizione di Intelligenza Artificiale (IA): L'IA è lo studio di "agenti intelligenti", che percepiscono il loro ambiente e producono azioni volte a massimizzare la probabilità di successo nel raggiungere i loro scopi.

L'IA è una disciplina situata all'incrocio tra informatica e psicologia, nata nella seconda metà del 20° secolo, che studia se e in che modo si possano riprodurre al computer i processi mentali più complessi.

La nascita dell'IA, sia come termine che come disciplina, viene fatta risalire ad un convegno fondativo nel 1956 al Dartmouth College (USA), organizzato da John McCarthy, Marvin Minsky, Nathaniel Rochester e Claude Shannon. La proposta di questo workshop conteneva già il termine "Intelligenza Artificiale", coniato da McCarthy.

Tuttavia, il concetto di macchine intelligenti è più antico. Alan Turing (1912-1954) aveva già posto le basi filosofiche per l'IA con il suo articolo del 1950 "Computing Machinery and Intelligence", dove si poneva la domanda: "Mi propongo di affrontare il problema se sia possibile per ciò che è meccanico manifestare un comportamento intelligente".

1.1.1 Il Test di Turing

Il test di Turing è un criterio per determinare se una macchina sia in grado di pensare. In questo test:

- Un valutatore umano intrattiene una conversazione (limitata a testo scritto) con un altro umano e una macchina
- Il valutatore non sa quale sia l'umano e quale la macchina
- Se il valutatore non è in grado di distinguere in modo affidabile l'umano dalla macchina, allora la macchina è considerata "intelligente"

Questo test pone l'accento non sulla definizione formale di intelligenza, ma sulla sua manifestazione comportamentale esterna, proponendo un approccio operativo all'intelligenza artificiale.

1.1.2 IA Debole e IA Forte

Intelligenza artificiale "stretta" (narrow artificial intelligence), nota anche come "IA debole":

- Si riferisce a qualsiasi intelligenza artificiale in grado di eguagliare o superare un essere umano in un compito strettamente definito e strutturato
- Esempi: riconoscimento di immagini, giocare a scacchi, diagnosi di malattie specifiche
- È lo stato attuale della tecnologia IA

Intelligenza artificiale "generale" (artificial general intelligence - AGI), nota anche come "IA forte":

- Dovrebbe consentire alle macchine di applicare conoscenze e abilità in diversi contesti, anche di tipo nuovo
- Ha capacità di comprensione, apprendimento e adattamento paragonabili a quelle umane
- È un obiettivo non ancora realizzato

1.2 Approcci all'Intelligenza Artificiale

Esistono due principali approcci filosofici e metodologici per sviluppare l'IA:

1.2.1 L'Approccio Logico-Simbolico

- La mente è vista come un computer che manipola simboli in modo algoritmico

- L'IA si realizza scrivendo un programma (intelligenza programmata)
- I processi simbolici sono indipendenti dallo specifico medium su cui vengono realizzati
- Pensare significa manipolare, attraverso regole, delle strutture simboliche di rappresentazione

Questo approccio, supportato da figure come Turing (1950) e Newell & Simon (1967), si basa sull'ipotesi del sistema simbolico fisico che afferma che "un sistema simbolico fisico possiede i mezzi necessari e sufficienti per l'azione intelligente generale."

I sistemi basati sulla conoscenza tipici di questo approccio includono:

- **Rappresentazioni simboliche strutturate** (base di conoscenza)
  - Ontologie: insiemi di concetti, assiomi e relazioni che descrivono un dominio di interesse
- **Sistema di regole** (motore inferenziale)

Limitazioni dell'approccio simbolico:

- La conoscenza sul dominio deve essere trasmessa da un esperto umano per essere codificata in modo esplicito
- Difficoltà a trasmettere conoscenze empiriche (informali o implicite)
- Il sistema elabora in modo deterministico

1.2.2 L'Approccio Neurale (Connessionismo)

- La mente emerge dalle dinamiche di reti neurali complesse in interazione con l'ambiente
- L'IA si realizza simulando reti neurali (intelligenza emergente)
- I processi mentali emergono da uno specifico medium: neuroni collegati in una rete
- Pensare è visto come l'attività di neuroni che formano complesse reti neurali

Questo approccio, sostenuto da ricercatori come McCulloch & Pitts (1943) e Hebb (1949), si concentra sulla simulazione di reti neurali ispirate al funzionamento del cervello umano.

1.3 Storia delle Reti Neurali

La storia delle reti neurali può essere suddivisa in diverse fasi:

1.3.1 Gli inizi (1943-1960)

- **Neurone artificiale** - McCulloch & Pitts, 1943: primo modello matematico del neurone
- **Regola di apprendimento sinaptico** - Hebb, 1949: prima teoria dell'apprendimento neurale
- **Percettrone** (Rosenblatt, 1958): primo modello di rete neurale con capacità di apprendimento

- **Regola delta** (Widrow & Hoff, 1960): algoritmo di apprendimento per minimizzare l'errore

1.3.2 Il declino (1969-1981)

- **Critiche al percettrone** - Minsky & Papert, 1969: libro che evidenziava le limitazioni del percettrone, soprattutto l'impossibilità di risolvere problemi non linearmente separabili
- Calo di interesse e finanziamenti per l'IA neurale (primo "inverno dell'IA")

1.3.3 Il rinascimento (1982-1990)

- **Memorie associative** – Hopfield, 1982: introduzione delle reti neurali ricorrenti
- **Apprendimento con "error backpropagation"** – Rumelhart, Hinton, 1985: algoritmo di apprendimento che supera le limitazioni del percettrone
- **Connessionismo** – Rumelhart, McClelland & PDP group, 1986: approccio che utilizza reti neurali per modellare i processi cognitivi

1.3.4 Nuovi sviluppi (1998-2005)

- **Apprendimento per rinforzo** - Sutton & Barto, 1998: formalizzazione dei principi dell'apprendimento per rinforzo
- **Apprendimento non supervisionato / generativo** – Hinton & Sejnowski, 1999: modelli per apprendere la struttura statistica dei dati senza supervisione

1.3.5 La rivoluzione del "deep learning" (2006-oggi)

- **Deep learning** – Hinton & Salakhutdinov, 2006: apprendimento efficace di reti neurali con molti strati
- **Deep convolutional neural networks**: architetture specializzate per l'elaborazione di immagini
- **Transformers**: architetture per l'elaborazione del linguaggio naturale

1.4 Applicazioni dell'IA

Le moderne applicazioni dell'IA includono:

- **Riconoscimento e/o generazione di immagini**: sistemi di computer vision, generatori di immagini (DALL-E, Midjourney)
- **Riconoscimento e/o generazione del parlato**: assistenti vocali, sistemi di dettatura
- **Elaborazione del linguaggio naturale e traduzione**: chatbot, traduttori automatici
- **Supporto alla decisione, sistemi di raccomandazione**: sistemi di suggerimento prodotti, sistemi di supporto clinico
- **Rappresentazione della conoscenza, ricerca e recupero di informazioni**: motori di ricerca, sistemi esperti

- **Sistemi di controllo**: guida assistita/autonoma, robotica, controllo di infrastrutture

I successi attuali si basano principalmente sull'IA neurale, in particolare sul deep learning, che ha beneficiato di:

- **Big data**: accesso a grandi quantità di dati etichettati
- **GPU computing**: enorme potenza di calcolo parallelo su schede di processori grafici

1.5 L'IA tra "hype" e "inverni"

La storia dell'IA è caratterizzata da cicli di entusiasmo eccessivo ("hype") seguiti da periodi di disillusione e calo di interesse/finanziamenti ("inverni dell'IA").

Esempi di previsioni eccessive:

- "In 3 to 8 years we will have a machine with a level of general intelligence comparable to that of an average human being" (Marvin Minsky, 1967)
- "Fully autonomous Tesla will drive across the country by the end of 2017" (Elon Musk, 2016)

L'IA ha attraversato due principali "inverni":

1. **Primo inverno dell'IA (1974-1980)**: dopo le critiche al percettrone e il fallimento dei sistemi di traduzione automatica
2. **Secondo inverno dell'IA (1987-1993)**: dopo il calo di interesse nei sistemi esperti

È importante mantenere aspettative realistiche per evitare un terzo "inverno dell'IA".

2. FONDAMENTI MATEMATICI

2.1 Introduzione ai Concetti Matematici per l'IA

La modellazione computazionale trae vantaggio dall'utilizzo di strumenti di ricerca precisi, sistematici e chiaramente definiti. Un avanzamento più rapido della conoscenza scientifica è supportato dall'adozione di un linguaggio formale (matematico), dove i concetti vengono definiti nel modo più esplicito e preciso possibile.

Una comprensione più profonda dei fondamenti matematici ci aiuta ad essere più critici e a comprendere meglio i punti di forza e i limiti dei vari metodi di IA.

2.2 Teoria dei Grafi

La teoria dei grafi è un potente formalismo matematico usato per descrivere e studiare reti di elementi che interagiscono tra loro.

2.2.1 Elementi Fondamentali

- **Nodi** ("vertici" o "unità"): definiscono le variabili da modellare
- **Archi** ("connessioni" o "spigoli"): definiscono quali variabili interagiscono direttamente tra di loro
- **Pesi**: definiscono la forza delle interazioni

2.2.2 Applicazioni della Teoria dei Grafi

La teoria dei grafi viene utilizzata per modellare:

- Atomi e molecole
- Proteine e reti genetiche
- Sistemi immunitari
- Specie animali ed ecosistemi
- Flussi di traffico
- Reti di telecomunicazione (es: Internet)
- Traffici commerciali
- Relazioni sociali
- Reti di neuroni
- E molto altro

2.2.3 Tipi di Grafi

Grafi direzionati:

- Codificano relazioni di parentela tra le variabili
- La direzionalità è spesso relata alla nozione di "causalità"
- Esempio: Il "canto del gallo" è causato dal "sorgere del sole", non viceversa

Grafi non direzionati:

- Codificano solo il "grado di affinità" (correlazione) tra variabili
- Non stabiliscono relazioni causa-effetto
- Esempio: Sappiamo che "felicità di Bob" e "felicità di Alice" sono correlate, ma non sappiamo quale causa l'altra

2.2.4 Topologia dei Grafi

La topologia di un grafo specifica la mappa di connettività su un piano bidimensionale. Alcune topologie comuni includono:

- **Grafo completamente connesso**: ogni nodo è collegato a tutti gli altri
- **Grafo ad anello**: i nodi formano un circuito chiuso
- **Grafo ad albero**: struttura gerarchica senza cicli

- **Grafo bipartito**: i nodi sono divisi in due insiemi, e gli archi collegano solo nodi di insiemi diversi

La topologia e la direzionalità del grafo definiscono l'architettura di una rete neurale.

2.3 Teoria della Probabilità

La teoria della probabilità è un formalismo matematico usato per descrivere e studiare i sistemi stocastici.

2.3.1 Perché Studiare Sistemi Stocastici?

- **Ignoranza**: spesso possiamo osservare un sistema solo parzialmente
  - Es: dati alcuni sintomi osservati, diagnosticare la malattia che li ha generati
- **Rumore nei dati**: spesso i dati contengono errori ed approssimazioni
  - Es: le misure sperimentali, ma anche le sensazioni stesse, sono rumorose
- **Sistemi intrinsecamente stocastici**: alcuni sistemi hanno una natura probabilistica
  - Es: fisica quantistica, sistemi complessi

2.3.2 Elementi Fondamentali

- **Variabile casuale**: variabile il cui valore è definito in base ad una qualche funzione di probabilità
  - Nota: "casuale" non significa necessariamente "impredicibile" o "totalmente random"
  - Una variabile casuale può seguire distribuzioni probabilistiche ben definite
- **Probabilità condizionata** $P(A|B)$: probabilità di osservare un certo comportamento di una o più variabili casuali, dato il valore di altre variabili
- **Probabilità congiunta** $P(A, B)$: probabilità dell'intersezione di due o più variabili casuali
- **Indipendenza statistica**: due variabili sono indipendenti se non si influenzano a vicenda. In tal caso:

$$P(A, B) = P(A) \cdot P(B)$$

$$P(A|B) = P(A)$$

2.3.3 Distribuzioni di Probabilità Comuni

- **Distribuzione uniforme**: tutti i valori hanno la stessa probabilità
- **Distribuzione normale (gaussiana)**: distribuzione a campana, caratterizzata da media e varianza
- **Distribuzione esponenziale**: descrive il tempo tra eventi in un processo di Poisson

2.3.4 Teorema di Bayes

Un teorema fondamentale della teoria della probabilità è il teorema di Bayes:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Dove:

- $P(A|B)$ è la probabilità a posteriori
- $P(A)$ è la probabilità a priori
- $P(B|A)$ è la verosimiglianza
- $P(B)$ è la probabilità marginale

2.3.5 La Prospettiva Bayesiana

- **Inferenza**: data una certa evidenza (variabili osservate), vogliamo trovare la distribuzione di probabilità delle variabili non osservate (ipotesi)
- **Apprendimento**: vogliamo trovare il set di parametri che meglio descrive un insieme di osservazioni (massima verosimiglianza)

La prospettiva bayesiana interpreta il valore di ciascuna variabile casuale come un grado di credibilità (degree of belief), che viene costantemente aggiornato in base all'evidenza corrente.

2.4 Algebra Lineare

L'algebra lineare è il ramo della matematica che studia i vettori, gli spazi vettoriali (o spazi lineari), le trasformazioni lineari e i sistemi di equazioni lineari.

2.4.1 Elementi Fondamentali

- **Spazio vettoriale**: insieme di entità chiamate vettori, che possono essere sommate tra di loro e moltiplicate (scalate) per numeri reali, chiamati scalari
- **Matrice**: insieme ordinato di elementi, rappresentato in forma tabulare, disponendo gli elementi per righe e colonne

2.4.2 Concetti Chiave

- **Iperpiano**: sottospazio di dimensione inferiore di uno $(n-1)$ rispetto allo spazio in cui è contenuto (n)
  - Es: In uno spazio 2D, gli iperpiani sono singole rette (1D)
  - Es: In uno spazio 3D, gli iperpiani sono i piani 2D
- **Trasformazione (mapping) lineare**: funzione fra due spazi vettoriali che preserva le operazioni di addizione tra vettori e moltiplicazione scalare
  - In senso intuitivo, un mapping lineare non altera la "linearità" di una curva
  - Può stirare e riscalare i vari punti, ma non può alterarne la curvatura

2.4.3 Operazioni Matriciali Fondamentali

- **Somma di matrici**: $(A + B)_{ij} = A_{ij} + B_{ij}$
- **Moltiplicazione per scalare**: $(cA)_{ij} = c \cdot A_{ij}$
- **Prodotto matriciale**: $(AB)_{ij} = \sum_k A_{ik} \cdot B_{kj}$
- **Trasposta**: $(A^T)_{ij} = A_{ji}$
- **Determinante**: misura quanto una trasformazione lineare espande o contrae lo spazio
- **Matrice inversa**: $A \cdot A^{-1} = A^{-1} \cdot A = I$ (matrice identità)

2.5 Analisi Matematica (Calculus)

L'analisi matematica studia come le funzioni cambiano nel tempo e nello spazio.

2.5.1 Elementi Fondamentali

- **Derivata di una funzione**: sensibilità al cambiamento di una certa quantità (variabile dipendente), la quale è determinata da un'altra quantità (variabile indipendente)
  - La derivata f' di una funzione f in un preciso punto corrisponde alla pendenza della retta tangente alla curva della funzione in quel punto
  - La funzione deve essere continua
- **Gradiente di una funzione**: generalizza il concetto di derivata a funzioni multivariate
  - Il gradiente di una funzione corrisponde al vettore le cui componenti sono le derivate parziali della funzione
  - $\nabla f(x, y, z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)$
- **Massimi e minimi di una funzione**: punti stazionari, dove la curva raggiunge il valore massimo (o minimo)
  - Se un punto corrisponde ad un massimo o ad un minimo, allora la derivata della funzione in quel punto è nulla
  - Questa proprietà è fondamentale per l'ottimizzazione nelle reti neurali

2.5.2 Discesa del Gradiente

La discesa del gradiente è una tecnica di ottimizzazione per trovare il minimo di una funzione. Il metodo consiste nel prendere passi proporzionali al negativo del gradiente della funzione nel punto corrente:

$$x_{n+1} = x_n - \eta \nabla f(x_n)$$

dove η è il tasso di apprendimento (learning rate).

Varianti della discesa del gradiente:

- **Batch gradient descent**: utilizza tutti i dati del training set per calcolare il gradiente

- **Stochastic gradient descent (SGD)**: utilizza un solo esempio alla volta per calcolare il gradiente
- **Mini-batch gradient descent**: utilizza un piccolo sottoinsieme di esempi per calcolare il gradiente

2.6 Sistemi Complessi Non-Lineari

I sistemi non-lineari sono caratterizzati da relazioni non proporzionali tra input e output.

2.6.1 Modelli Lineari vs Non-Lineari

- **Modelli lineari**:
  - Sono semplici da analizzare matematicamente
  - Hanno una capacità espressiva limitata
  - La traiettoria può essere predetta con precisione (es: pendolo semplice)
- **Modelli non-lineari**:
  - Sono molto più espressivi e possono rappresentare relazioni complesse
  - Sono più difficili da analizzare
  - Possono mostrare comportamenti caotici (es: pendolo doppio)

2.6.2 Il Concetto di Caos

Nei sistemi caotici, piccole variazioni nelle condizioni iniziali producono grandi variazioni nel comportamento a lungo termine, rendendo le previsioni a lungo termine impossibili.

Questa sensibilità alle condizioni iniziali è nota come "effetto farfalla", dal titolo di un articolo del meteorologo Edward Lorenz: "Il battito d'ali di una farfalla in Brasile può provocare un tornado in Texas".

2.7 Algoritmi di Apprendimento Automatico

2.7.1 Definizione e Concetto

Gli algoritmi di apprendimento automatico (machine learning) sono progettati per fare in modo che l'algoritmo stesso possa modificare la propria "struttura" con l'esperienza.

- Il comportamento dell'algoritmo dipende dal valore di alcuni suoi stati interni
- L'algoritmo ha la capacità di modificare in modo autonomo tali stati in base al tipo di esperienza (input) che riceve
- Nel caso di una rete neurale artificiale, il sistema modifica la forza delle connessioni sinaptiche in base alle informazioni rilevate in un particolare insieme di esempi di addestramento (training set)

2.7.2 Ambienti di Sviluppo per Machine Learning

Esistono vari strumenti e linguaggi per implementare algoritmi di machine learning:

- **Python**: linguaggio più utilizzato per il machine learning
  - Librerie: TensorFlow, PyTorch, scikit-learn, Keras
- **MATLAB**: ambiente di calcolo numerico con linguaggio proprietario
- **Java**: utilizzato per applicazioni enterprise di machine learning
- **R**: linguaggio specializzato per statistica e data science

3. COMPUTAZIONE NEURALE

3.1 Definizione e Concetti di Base

La computazione neurale (neural computing / neural computation) è l'elaborazione dell'informazione eseguita da reti di neuroni, biologici o artificiali. Rappresenta un modello alternativo a quello della computazione digitale.

Le reti neurali biologiche sono formate da neuroni che si influenzano attraverso le connessioni (sinapsi) che li collegano:

- Ogni neurone rileva un certo insieme di condizioni e segnala ciò che ha rilevato attraverso la sua frequenza di scarica
- I neuroni possono ricevere segnali da altri neuroni e formare strati di rilevatori più complessi
- Le interazioni tra neuroni sono adattive e si modificano attraverso l'apprendimento

3.1.1 Il Modello del "Rilevatore"

Ogni neurone "rileva" un qualche insieme di condizioni. La rappresentazione è ciò che viene rilevato.

Confronto tra Computer e Rilevatori:

Computer	Rilevatori
Memoria ed elaborazione separate, uso generico	Memoria ed elaborazione integrate, specializzate
Operazioni logiche, aritmetiche	Detezione/rilevazione (accumulo di evidenza, valutazione, comunicazione)
Elaborazione complessa tramite sequenze arbitrarie di operazioni concatenate in un programma	Elaborazione complessa tramite gruppi di rilevatori altamente "sintonizzati", organizzati in strati

3.1.2 Codifica dell'Informazione nel Neurone Biologico

Il neurone è "sintonizzato" su uno "stimolo preferito", che quando è presente produce la risposta più forte. Il campo recettivo del neurone è quella porzione di spazio sensoriale (ad es. posizione sulla retina) che attiva la risposta del neurone.

Gli studi di Hubel & Wiesel (1959) con registrazioni da aree visive primarie (V1) hanno dimostrato questa selettività:

- Neuroni selettivi per l'orientamento degli stimoli visivi
- Neuroni con campi recettivi strutturati (aree ON e OFF)
- Risposta massima per stimoli specifici (es: barre con un certo orientamento)

Campo recettivo di un neurone della corteccia visiva primaria (V1):

1. Il campo recettivo ha un'area centrale di eccitazione (ON) circondata da aree di inibizione (OFF)
2. La risposta del neurone (frequenza di scarica) è massima per stimoli con orientamento specifico
3. La risposta varia in base al tipo di stimolo: luce puntiforme, barra orientata, luce diffusa, ecc.

3.1.3 Elaborazione Gerarchica

L'elaborazione gerarchica è un principio generale di organizzazione neurale, dove la codifica dell'informazione diventa più complessa ed astratta passando a livelli più elevati (profondi) della gerarchia.

Felleman & Van Essen (1991) e Riesenhuber & Poggio (1999) hanno mappato questa gerarchia nel sistema visivo:

- Neuroni nella corteccia visiva primaria (V1) rispondono a caratteristiche semplici come bordi e orientamento
- Neuroni in aree superiori (V2, V4) rispondono a forme più complesse
- Neuroni in aree ancora più elevate (IT, infero-temporale) possono rispondere a oggetti interi o volti

Questa organizzazione gerarchica è la principale ispirazione per il deep learning.

3.2 Reti Neurali Artificiali

3.2.1 Definizione e Caratteristiche

Le reti neurali artificiali sono sistemi di elaborazione:

- Caratterizzati dalla capacità di apprendere compiti complessi
- Ispirati dal funzionamento dei sistemi biologici
- L'obiettivo non è di riprodurne tutta la complessità ma di catturarne i principi di base

Una rete neurale artificiale ha un'architettura ad elaborazione distribuita e parallela che utilizza:

- Semplici unità di elaborazione
- Un elevato grado di interconnessione
- Semplici messaggi numerici scalari (segnali "analogici")
- Interazioni adattive tra elementi

3.2.2 Proprietà delle Reti Neurali Artificiali

- Apprendono dall'esperienza ma rispondono anche in situazioni nuove
- Non necessitano di conoscenze esplicite sul problema
- Si adattano a situazioni caotiche e/o contraddittorie ("rumore" nei dati)

3.2.3 Il Neurone Artificiale

Un neurone formale (unità di elaborazione o nodo) è un modello matematico che cerca di catturare gli aspetti fondamentali del funzionamento neuronale:

- I neuroni hanno connessioni con altri neuroni dai quali ricevono segnali
- Ogni connessione ha un peso che indica la forza della connessione e può essere positivo (eccitatorio) o negativo (inibitorio)
- L'input che un neurone riceve da ciascuno dei neuroni cui è connesso si calcola moltiplicando il segnale proveniente da quel neurone per il peso sulla connessione
- L'input totale del neurone è la sommatoria delle attivazioni che il neurone riceve dagli altri neuroni
- Lo stato di attivazione finale viene calcolato attraverso una funzione di attivazione o di output del neurone
- L'output del neurone viene inviato ai neuroni "a valle"

Matematicamente, il funzionamento di un neurone artificiale può essere descritto come:

$$\mathrm{net}_i = \sum_j w_{ij} x_j + b_i$$

$$o_i = f(\mathrm{net}_i)$$

dove:

- x_j: segnale in arrivo dal neurone j
- w_{ij}: peso sinaptico della connessione tra il neurone j ed il neurone ricevente i
- b_i: soglia o "bias" del neurone i
- net_i: input totale al neurone i
- o_i: attivazione del neurone i (altre notazioni: y o a) ed output verso altri neuroni
- f: funzione di attivazione

3.2.4 Funzioni di Attivazione

Le funzioni di attivazione più comuni includono:

- **Funzione a gradino (step function)**:

$$f(x) = \{1 \quad \text{se } x \geq 0 \; 0 \quad \text{se } x < 0$$

- **Funzione sigmoide**:

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

- **Tangente iperbolica**:

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- **ReLU (Rectified Linear Unit)**:

$$f(x) = \max(0, x)$$

- **Leaky ReLU**:

$$f(x) = \{x \quad \text{se } x \geq 0 \; \alpha x \quad \text{se } x < 0$$

  dove α è un piccolo valore positivo (es: 0.01)
- **Softmax** (per output multi-classe):

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

3.2.5 Architetture di Reti Neurali

L'architettura della rete identifica:

L'organizzazione in gruppi o strati dei neuroni (topologia della rete):

- Unità che ricevono input direttamente dall'"ambiente" formano lo strato di input
- Unità che producono l'output finale della rete formano lo strato di output
- Unità che non si trovano in contatto diretto con input o output sono chiamate unità nascoste (hidden)
- Con $n \geq 2$ strati nascosti parleremo di rete profonda (deep network)

Il modo in cui i neuroni sono collegati tra di loro (schema di connettività):

- **Reti feed-forward**: ci sono solo connessioni unidirezionali da unità di input a unità nascoste a unità di output (bottom-up)
- **Reti ricorrenti**: ci sono connessioni bidirezionali, in cui l'attivazione può propagarsi all'indietro (top-down o feedback)

- **Reti interamente ricorrenti**: come le reti ricorrenti, ma ci sono anche connessioni intra-strato (laterali)

3.2.6 Tipi di Architetture di Reti Neurali

1. **Associatore di pattern**:
   - Solo unità di input e di output
   - Connessioni unidirezionali
2. **Rete multi-strato feed-forward**:
   - Simile all'associatore, ma contiene uno o più strati di unità nascoste
   - Connessioni unidirezionali
3. **Rete ricorrente**:
   - L'attivazione può propagarsi all'indietro verso gli strati nascosti e di input
   - Connessioni bidirezionali (feedback)
4. **Rete completamente ricorrente**:
   - Simile ad una rete ricorrente ma ci sono anche connessioni intra-strato (laterali)
   - È l'architettura più generale

3.3 Reti Neurali Biologiche vs. Reti Neurali Artificiali

Dire che le reti neurali artificiali sono ispirate a quelle biologiche non significa riprodurre il neurone biologico e le reti di neuroni in tutta la loro complessità, bensì catturarne i principi base di funzionamento.

Cosa manca nel tipico neurone artificiale:

- Organizzazione spaziale dei contatti sinaptici
- Differenziazione tra neuroni eccitatori ed inibitori
- Tipi diversi di sinapsi (neurotrasmettitori)
- Dinamica del neurone con potenziali di azione (ma esistono "spiking neural networks")

Cosa manca nella tipica rete artificiale:

- Struttura laminare e organizzazione colonnare
- Organizzazione in mappe topografiche
- Differenza di scala (nel cervello ci sono circa 10^11 neuroni, e un neurone comunica in media con 10^4 altri neuroni)

4. APPRENDIMENTO E MEMORIA NELLE RETI NEURALI

4.1 Paradigmi di Apprendimento

L'apprendimento automatico (machine learning) si riferisce a metodi e algoritmi che permettono ad un software di apprendere dall'esperienza.

4.1.1 Tipologie di Apprendimento

Immaginiamo un sistema (biologico o artificiale) che riceve una serie di input sensoriali: $x_1, x_2, x_3, x_4, \ldots$

Apprendimento supervisionato:

- Viene fornito anche l'output desiderato $y_1, y_2, \ldots$
- Lo scopo è di imparare a produrre l'output corretto dato un nuovo input
- Esempi: classificazione, regressione

Apprendimento non supervisionato:

- Lo scopo è di costruire rappresentazioni dell'input scoprendone le proprietà più importanti ed informative
- Queste rappresentazioni possono essere in seguito utilizzate per il ragionamento, la decisione, la comunicazione, etc.
- Esempi: clustering, riduzione della dimensionalità, apprendimento di rappresentazioni

Apprendimento per rinforzo:

- Il sistema produce azioni che hanno un effetto sul mondo, e riceve rinforzi (o punizioni) $r_1, r_2, \ldots$
- Lo scopo è di imparare ad agire in un modo che massimizza il rinforzo nel lungo termine
- Esempi: controllo di robot, giochi, guida autonoma

4.1.2 Tassonomia dei Metodi di Apprendimento

Machine Learning Convenzionale (shallow learning):

- Supervisionato (classificazione, regressione)
- Non supervisionato (riduzione della dimensionalità, clustering)
- Per rinforzo

Deep Learning:

- Discriminativo (supervised)
- Generativo (unsupervised)
- Deep RL (reinforcement learning)

4.2 Apprendimento nelle Reti Neurali

4.2.1 Principi Generali

L'apprendimento in una rete neurale consiste nel trovare l'insieme di pesi delle connessioni che permette alla rete di produrre la risposta appropriata ad un certo input.

In generale, il processo di apprendimento include:

- Valori iniziali dei pesi sinaptici assegnati in modo casuale (es. tra -0.1 e +0.1)
- Presentazione ripetuta dei pattern di addestramento
  - Apprendimento supervisionato: input + target (output desiderato)
  - Apprendimento non-supervisionato: solo input
- Modifica dei pesi, ovvero il calcolo di Δw rispetto a w precedente
- Aggiornamento dei pesi può avvenire dopo ogni pattern (on-line learning) oppure dopo ogni epoca di apprendimento (batch learning)
- Utilizzo di un learning rate (tasso di apprendimento) η per controllare la velocità dell'apprendimento:

$$w_{ij}^t = w_{ij}^{t-1} + \eta \Delta w_{ij}^t$$

4.2.2 La Regola di Hebb

La forma più semplice di apprendimento è la regola di Hebb:

> "Se due neuroni collegati tra loro sono contemporaneamente attivi, l'efficacia sinaptica della connessione viene aumentata."

(D. Hebb, The organization of behavior, 1949)

L'apprendimento hebbiano è biologicamente plausibile e corrisponde al fenomeno del potenziamento a lungo termine (Long Term Potentiation; LTP) scoperto da Kandel e colleghi nel 1964.

Formalmente, se vogliamo associare un pattern di input x con un pattern di output y, la regola di Hebb è:

$$\Delta w_{ij} = \eta x_i y_j$$

Nella sua forma originale, la regola presenta alcune limitazioni:

- Può solo rinforzare le connessioni (non può indebolirle)
- I valori dei pesi non hanno un limite superiore

Varianti della regola di Hebb (come la regola della covarianza per le reti di Hopfield) superano questi limiti.

4.2.3 Algoritmi di Apprendimento per Reti Neurali

Esistono molti algoritmi di apprendimento diversi per reti neurali, tra cui:

- **Regola di Hebb**: apprendimento associativo semplice

- **Algoritmo del percettrone**: per reti a singolo strato
- **Regola delta (o di Widrow-Hoff)**: minimizzazione dell'errore quadratico
- **Backpropagation**: per reti multi-strato
- **Contrastive divergence**: per macchine di Boltzmann ristrette
- **Algoritmi genetici**: per ottimizzazione dei pesi

La scelta dell'algoritmo dipende dal tipo di rete neurale e dal compito di apprendimento.

4.3 Reti Neurali e Memoria

4.3.1 Tipi di Memoria nelle Reti Neurali

Una rete neurale non contiene una sorta di magazzino di memoria separato, ma rappresenta l'informazione in modi diversi:

Memoria a lungo termine:

- I pesi delle connessioni (pesi sinaptici) rappresentano le conoscenze a lungo termine
- Non si cancellano e si modificano (gradualmente) solo se c'è ulteriore apprendimento

Memoria a breve termine:

- L'attivazione dei neuroni è un fenomeno temporaneo
- È specifica per lo stimolo presentato e si esaurisce con la sua scomparsa
- Se l'attivazione non cessa bruscamente, può influenzare l'elaborazione dello stimolo successivo (processo alla base dei fenomeni di priming)

Memoria di lavoro:

- Alcuni compiti cognitivi richiedono di ricordare per breve tempo informazioni che non sono più presenti
- In una rete neurale questo si può realizzare mantenendo attivi determinati neuroni anche quando l'input non è più presente
- Nelle reti ricorrenti, questa memoria può essere rappresentata dallo stato interno della rete

4.3.2 Il Problema dell'Interferenza

Il compito di apprendimento associativo AB-AC porta alla luce il problema dell'interferenza nella memoria:

- Vi sono due liste di coppie di parole
    - Lista AB: ogni coppia è costituita da una parola del gruppo A e da una parola del gruppo B

    - Lista AC: ogni coppia è costituita da una parola del gruppo A e da una parola del gruppo C
- I partecipanti dopo aver studiato la lista AB devono rievocare la parola B appropriata per ogni parola A
- Poi studiano la lista AC e vengono interrogati su entrambe le liste

I risultati mostrano un'interferenza delle associazioni apprese successivamente (AC) su quelle apprese per prime (AB).

4.3.3 Interferenza Catastrofica nelle Reti Neurali

Se una rete neurale viene sottoposta al compito di apprendimento associativo AB-AC si ottiene un effetto di interferenza ancora maggiore che negli esseri umani. Questo fenomeno è stato chiamato di "interferenza catastrofica" (catastrophic forgetting).

Due fattori determinano l'interferenza in una rete neurale:

- Il grado di sovrapposizione delle rappresentazioni (che tuttavia è utile perché permette l'integrazione e la generalizzazione delle conoscenze)
- Il tasso di apprendimento elevato (che è ovviamente utile per un apprendimento rapido)

4.3.4 Due Sistemi Complementari di Apprendimento e Memoria

Il cervello umano ha risolto questo problema evolvendo due sistemi di apprendimento separati e complementari:

Ippocampo:

- Specializzato nell'apprendimento rapido e non soggetto a interferenza
- Utilizza rappresentazioni sparse (pochi neuroni attivi contemporaneamente)
- Permette la memorizzazione rapida di eventi specifici

Neocorteccia:

- Apprende lentamente e integra gradualmente le esperienze
- Estrae le conoscenze generali sul mondo
- Utilizza rappresentazioni distribuite (molti neuroni attivi con pattern specifici)
- Permette la generalizzazione e il trasferimento dell'apprendimento

Questo modello a "sistema duale" di memoria è stato proposto da McClelland, McNaughton e O'Reilly (1995) e ha ispirato molti studi successivi sulle reti neurali.

5. APPRENDIMENTO SUPERVISIONATO

5.1 Introduzione all'Apprendimento Supervisionato

L'apprendimento supervisionato è un paradigma di machine learning in cui la rete impara ad associare un dato input ad uno specifico output.

Caratteristiche principali:

- L'output "desiderato" (ovvero la risposta corretta) deve essere fornito esplicitamente alla rete durante l'apprendimento
- Tutti gli esempi per l'apprendimento devono essere "etichettati" (labeled data) con l'output desiderato

Applicazioni:

- Classificazione: associare input a categorie discrete (es: riconoscimento di immagini, spam detection)
- Regressione: associare input a valori continui (es: previsione di prezzi, stima di età)

Relazione con metodi statistici:

- Modello lineare generalizzato (GLM)
- Regressione
- Analisi discriminante

5.2 La Correzione dell'Errore

5.2.1 Elementi Fondamentali

Training set: insieme di esempi (patterns) su cui addestrare la rete. Per ogni esempio possiamo definire:

- **Input (x)**: la configurazione dei valori di input (in statistica: variabili indipendenti; predittori)
- **Output (y)**: valore/i che rappresentano la risposta della rete (in statistica: variabile dipendente)
- **Target (t)**: la risposta desiderata/corretta da associare all'input

5.2.2 Il Processo di Addestramento

1. Presentazione di un esempio (valori di input)
2. Calcolo output sulla base dei parametri attuali (pesi)
3. Confronto tra output e target per determinare lo scostamento / errore
4. Modifica dei parametri (ottimizzazione) per diminuire lo scostamento
5. Ripetizione del processo per tutti gli esempi del training set

5.2.3 Testing

Verifica delle prestazioni, inclusa la capacità di generalizzazione su dati non utilizzati per l'addestramento (test set).

5.3 Il Percettrone

Il percettrone è il primo modello di rete neurale con pesi sinaptici modificabili da un algoritmo di apprendimento, sviluppato da Frank Rosenblatt (psicologo americano) nel 1958.

5.3.1 Struttura e Funzionamento

La rete ha N input che codificano l'esempio presentato con valori x_i ed un singolo neurone di output che codifica la risposta in modo bipolare (o binario).

L'output y della rete è:

$$y = \begin{cases} 1 & \text{se } \sum_{i=1}^{N} w_i x_i - \theta \geq 0 \\ -1 & \text{altrimenti} \end{cases}$$

dove w_i è il peso della connessione dal neurone x_i e θ è la soglia del neurone.

Possiamo sostituire la soglia con un input fisso $x_0 = 1$ e il corrispondente peso w_0, chiamato bias, e calcolarne il valore come per qualsiasi altro peso durante l'apprendimento:

$$y = \begin{cases} 1 & \text{se } \sum_{i=0}^{N} w_i x_i \geq 0 \\ -1 & \text{altrimenti} \end{cases}$$

5.3.2 L'Apprendimento nel Percettrone

Per ogni esempio presentato, l'output y viene confrontato con la risposta desiderata (target) t che codifica la classe di appartenenza:

- Se $y = t$ (output corretto): si procede al prossimo esempio, senza fare alcuna modifica ai pesi
- Se $y \neq t$ (errore): i pesi sinaptici vengono modificati con:

$$\Delta w_i = \eta t x_i$$

dove η è il learning rate (tasso di apprendimento).

Si noti che i pesi delle connessioni da cui è arrivato input aumentano se il target è [+1] e diminuiscono se il target è [-1]. Nel caso di target binario ([0/1]) è sufficiente sostituire t con $(t - y)$.

5.3.3 Teorema di Convergenza del Percettrone

Per un qualunque problema linearmente separabile, il percettrone troverà la soluzione in un numero finito di passi.

Questo teorema garantisce che, se esiste una linea (o iperpiano in dimensioni superiori) che separa perfettamente le classi, il percettrone la troverà.

5.3.4 Limitazioni del Percettrone

1. **Molteplicità di soluzioni**: ci sono molte linee possibili che separano le due classi, e la soluzione trovata dipende dai valori iniziali dei pesi e dall'ordine di presentazione degli esempi
2. **Separabilità lineare**: il percettrone può risolvere solo problemi linearmente separabili; non può risolvere problemi come lo XOR (OR esclusivo)

5.4 Problemi di Separabilità Lineare

5.4.1 Problemi Linearmente Separabili

Un problema è linearmente separabile se esiste un iperpiano che separa gli esempi di classi diverse. Esempi:

- OR logico
- AND logico
- Problemi di classificazione semplici con classi ben separate

5.4.2 Problemi Non Linearmente Separabili

Un problema non è linearmente separabile se non esiste un iperpiano che separa gli esempi di classi diverse. Esempi:

- XOR logico
- Problemi di classificazione complessi con classi intrecciate

5.4.3 XOR: Un Problema Non Linearmente Separabile

La tabella di verità dello XOR:

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

Questo problema non può essere risolto da un percettrone, poiché non esiste una singola linea che possa separare i punti (0,0) e (1,1) dai punti (0,1) e (1,0) nel piano.

5.5 Associatore Lineare di Configurazioni

Un associatore di configurazioni è simile ad un percettrone ma i neuroni di output utilizzano una funzione di attivazione continua (come la sigmoide). L'output dei neuroni è quindi continuo (nel range 0-1). Questo permette di quantificare l'errore (confronto con errore si/no nel percettrone).

5.5.1 Struttura e Funzionamento

L'output del neurone è:

$$y_i = f\left(\sum_j w_{ij} x_j\right)$$

dove f è la funzione di attivazione, tipicamente la sigmoide:

$$f(net) = \frac{1}{1 + e^{-net}}$$

5.5.2 La Regola Delta

La regola delta è applicabile quando le unità di output hanno una funzione di attivazione continua e differenziabile (come la sigmoide).

Permette di descrivere la prestazione con una funzione che misura l'errore della rete - funzione di errore o funzione di costo - che si basa sullo scarto quadratico medio tra risposta desiderata ed output effettivo:

$$E(W) = \frac{1}{2} \sum_\mu \sum_i (t_i^\mu - y_i^\mu)^2$$

L'apprendimento consiste nel minimizzare la funzione di costo E, che dipende unicamente dal valore delle connessioni sinaptiche W. Quindi si modificano i pesi nella direzione opposta al gradiente della funzione stessa (discesa del gradiente):

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

5.5.3 Forma Finale della Regola Delta

La forma finale della regola dipende dal tipo di funzione di attivazione (perché va calcolato il valore della derivata: $f'(net)$).

Nel caso di una funzione lineare, otteniamo che il cambiamento dei pesi è dato semplicemente dalla differenza tra target e output moltiplicata per l'attività presinaptica:

$$\Delta w_{ij} = \eta (t_i - y_i) x_j$$

Per la funzione sigmoide avremo:

$$\Delta w_{ij} = \eta(t_i - y_i)y_i(1 - y_i)x_j$$

5.5.4 Apprendimento con la Regola Delta

1. Viene presentata alla rete una configurazione di input
2. L'attivazione fluisce alle unità di output
3. Viene calcolata l'attivazione delle unità di output
4. La configurazione così ottenuta viene confrontata con la configurazione di output desiderata
5. Viene calcolata la discrepanza tra le due configurazioni (segnale di errore)
6. I pesi sulle connessioni vengono modificati in modo da ridurre l'errore

La procedura viene eseguita per tutti gli esempi che formano il training set (epoca di apprendimento) ed ulteriormente ripetuta (eseguendo più epoche) fino a quando l'errore arriva a 0 oppure la "discesa" dell'errore si arresta.

NB: La regola delta è matematicamente equivalente alla regola di Rescorla-Wagner per il condizionamento classico.

5.6 Reti Multistrato

5.6.1 Perché Servono le Reti Multistrato?

I problemi caratterizzati da inseparabilità lineare (es. XOR) possono essere risolti da reti neurali multistrato (multi-layer perceptrons), ovvero con uno o più strati intermedi di neuroni nascosti (hidden layers) che utilizzano una funzione di attivazione non-lineare.

5.6.2 Teorema di Approssimazione Universale

La rete multistrato è un approssimatore universale: una rete neurale con almeno uno strato nascosto composto da un numero appropriato di neuroni con funzione di attivazione non-lineare può, in linea di principio, approssimare qualunque funzione X→Y (input-output).

5.6.3 Backpropagation

L'algoritmo di apprendimento noto come back-propagation (Rumelhart, Hinton & Williams, 1986) è una estensione della regola delta (regola delta generalizzata) che permette di addestrare reti multi-strato.

Architettura:

- Qualsiasi numero di strati nascosti (almeno uno)
- Qualsiasi connettività (anche reti ricorrenti, utilizzando una variante dell'algoritmo)

Problema risolto:

- Come calcolare l'errore per le unità nascoste tramite la propagazione all'indietro dell'errore (error back-propagation) calcolato per le unità di output attraverso le stesse connessioni che servono per la propagazione dell'attivazione

Nota sulla plausibilità biologica:

- La propagazione all'indietro dell'errore rende l'algoritmo implausibile dal punto di vista biologico

5.6.4 Propagazione in Avanti

Nella fase di propagazione in avanti (forward pass):

1. Gli input sono presentati alla rete
2. Ogni neurone calcola la somma pesata dei suoi input
3. Ogni neurone applica la funzione di attivazione alla somma pesata
4. Il risultato viene propagato ai neuroni dello strato successivo
5. Il processo continua fino allo strato di output

Per un neurone nascosto h_j:

$$h_j = f\left(\sum_k v_{jk} x_k\right)$$

Per un neurone di output y_i:

$$y_i = f\left(\sum_j w_{ij} h_j\right)$$

5.6.5 Error Backpropagation

Nella fase di propagazione all'indietro (backward pass):

1. Si calcola l'errore per le unità di output: differenza tra output desiderato e output effettivo
2. Si propaga questo errore all'indietro per calcolare l'errore "virtuale" delle unità nascoste
3. Si aggiornano i pesi di tutte le connessioni in base all'errore calcolato

Regola delta generalizzata:

Per i pesi tra strato nascosto e output:

$$\Delta w_{ij} = \eta \delta_i h_j$$

dove $\delta_i = (t_i - y_i) f'(net_i)$

Per i pesi tra input e strato nascosto:

$$\Delta v_{jk} = \eta \delta_j x_k$$

dove $\delta_j = f'(net_j) \sum_i \delta_i w_{ij}$

5.6.6 Apprendimento Batch vs. Online

Batch:

- Discesa del gradiente della funzione dell'errore globale
- I pesi vengono cambiati in base al gradiente calcolato attraverso tutti i pattern
- Più stabile, ma può essere lento e può rimanere intrappolato in minimi locali

Online o mini-batch:

- Discesa stocastica del gradiente
- I pesi vengono cambiati in base al gradiente della funzione dell'errore parziale, calcolato per un singolo esempio (o un piccolo numero di esempi per mini-batch)
- Meno stabile, ma può essere più veloce e può sfuggire a minimi locali

5.6.7 Tasso di Apprendimento

Il tasso di apprendimento η controlla la grandezza dei cambiamenti dei pesi:

Constante: $\eta = (0.01 \dots 0.5)$

- Piccolo: lento + maggiore probabilità di rimanere in minimi locali
- Grande: veloce + può essere impreciso e instabile

Variabile: $\eta = 0.5/\log(10 + k)$

- Dove k è il numero di epoca
- Permette un apprendimento più rapido all'inizio e più fine alla fine

5.6.8 Momentum

Il "momento" aggiunge all'aggiornamento del peso sinaptico una frazione del precedente cambiamento di valore:

$$\Delta w_{ij}^t = \alpha \Delta w_{ij}^{t-1} + \eta \delta_i x_j$$

dove $\alpha = 0.7 - 0.9$

Quando il gradiente dell'errore ha la stessa direzione, il momento aumenta la grandezza del passo che viene fatto (verso il minimo). Quando il gradiente cambia direzione, il momento affievolisce il cambiamento.

5.7 Generalizzazione

5.7.1 Definizione

La generalizzazione è la capacità di utilizzare in modo appropriato la conoscenza sul dominio quando si incontrano nuovi esempi del problema.

5.7.2 Condizioni per una Buona Generalizzazione

Condizioni necessarie (ma non sufficienti) per ottenere una buona generalizzazione:

- Le variabili di input contengono sufficienti informazioni relative al target, in modo che esista una funzione matematica che lega l'output corretto agli input con un determinato grado di accuratezza
- Gli esempi per l'addestramento sono in numero sufficientemente grande e sono un campione rappresentativo dell'insieme di casi a cui si vuole generalizzare (popolazione)

5.7.3 Interpolazione vs. Estrapolazione

Le reti multistrato possono apprendere ad "approssimare" qualunque funzione che lega l'input all'output, ma è utile distinguere tra:

- **Interpolazione**: predire output per input che si trovano "tra" i punti del training set (normalmente possibile)
- **Estrapolazione**: predire output per input che si trovano "fuori" dal range del training set (difficile e a volte impossibile)

5.8 Generalizzazione vs. Overfitting

5.8.1 Il Problema dell'Overfitting

Generalizzazione: la produzione di una risposta appropriata a pattern di input non utilizzati per l'addestramento, ovvero un test set indipendente dal training set.

Overfitting: si verifica quando continua a migliorare la prestazione sui pattern di addestramento (sul training set) ma peggiora la prestazione in termini di generalizzazione (sul test set indipendente).

L'overfitting può verificarsi quando:

- La relazione X-Y non è regolare (ha tante eccezioni)
- I dati contengono rumore
- La rete apprende anche il "rumore" nei dati usati per addestramento

5.8.2 Come Evitare l'Overfitting

Metodi per evitare l'overfitting (e quindi migliorare la generalizzazione):

- **Utilizzare reti neurali non troppo potenti**: permette di apprendere le regolarità statistiche nei dati piuttosto che "memorizzare" i pattern di training

    - Limitare il numero di unità nascoste
    - Verificare che lo strato nascosto sia necessario
- **Early stopping**: utilizzare un set di pattern (validation set) solo per la verifica di overfitting durante l'apprendimento e fermarlo prima che l'errore sul validation set inizi ad aumentare
- **Weight decay**: se i pesi hanno una tendenza spontanea a diminuire, i pesi più deboli (non rinforzati dall'apprendimento) tenderanno al valore zero
    - $\Delta w_{ij} = \eta \delta_i x_j - 0.001 w_{ij}$
    - Meno pesi = meno parametri. Questo impedisce di apprendere rumore nei dati
- **Dropout**: durante l'addestramento, a ogni iterazione, un sottoinsieme casuale di neuroni viene "spento" (tasso di dropout, es: 0.5)
    - Costringe la rete a sviluppare rappresentazioni ridondanti e robuste
    - Equivale a fare un'approssimazione dell'addestramento di un insieme di reti
- **Regolarizzazione L1 e L2**: aggiungere termini alla funzione di costo che penalizzano pesi grandi
    - L1: $\lambda \sum_i |w_i|$ (tende a produrre pesi sparsi)
    - L2: $\lambda \sum_i w_i^2$ (tende a distribuire i pesi)

Nota: In generale, i metodi per prevenire eccessivo adattamento (overfitting) sono chiamati metodi di regolarizzazione.

5.9 Training vs. Testing

5.9.1 Suddivisione dei Dati

- **Training set**: insieme di esempi (pattern) per l'addestramento. Sono utilizzati dall'algoritmo di apprendimento per trovare i valori dei pesi delle connessioni.
- **Validation set**: insieme di esempi utilizzati per ottimizzare parametri di apprendimento (detti anche iper-parametri, come learning rate, momentum, weight decay), il numero di unità nascoste, e per decidere quando fermare l'apprendimento (early stopping).
- **Test set**: insieme di esempi utilizzati per valutare la performance finale del modello.

Perché utilizzare set diversi per validation e test? Il set di validazione viene utilizzato per selezionare il modello migliore, quindi l'accuratezza sul validation set ha un bias (è sovrastimata).

5.9.2 Ripartizione degli Esempi

Come ripartire gli esempi? Utilizzare la maggiore quantità di dati possibile per l'addestramento. Se ci sono molti esempi, una possibile divisione è 50-25-25 (in percentuali).

5.9.3 Cross-validazione

Se i dati non sono sufficienti per ripartirli in insiemi separati di training e test, è essenziale massimizzare il numero di esempi di training utilizzando la tecnica di cross-validazione.

k-fold cross-validation:

- Il dataset totale viene diviso in k parti di uguale numerosità (spesso k=10)
- Ad ogni ciclo di cross-validazione, la k-esima parte del dataset viene esclusa dal training per essere utilizzata come test
- La performance finale è data dalla media tra i risultati di ogni ciclo (quindi tra tutte le k ripartizioni)

5.10 Valutazione della Performance

5.10.1 Principi Generali

- La valutazione della performance di una rete neurale va fatta sul test set
- Esistono molte metriche di performance, anche specifiche per dominio
- La distinzione principale tra metriche di performance è relativa a compiti di regressione vs classificazione

5.10.2 Metriche per Compiti di Regressione

Un compito di regressione implica uno o più output continui. La prestazione si valuta quindi in termini di "distanza" tra output effettivo ed output desiderato:

- **Errore quadratico medio (Mean Squared Error, MSE)**: $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- **Radice dell'errore quadratico medio (Root Mean Squared Error, RMSE)**: $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
- **Errore assoluto medio (Mean Absolute Error, MAE)**: $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
- **Proporzione di varianza spiegata (R^2)**: $1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$

Con una singola variabile di output è utile un grafico a dispersione che mostra le predizioni vs valori target.

5.10.3 Metriche per Compiti di Classificazione

Un compito di classificazione implica output binari o categorici. Una valutazione in termini di accuratezza (% risposte corrette) non è sufficiente per stabilire la qualità di un classificatore.

Matrice di confusione:

- Una tabella che mostra le predizioni del modello vs le classi reali
- Elementi della matrice di confusione:
  - TN (True Negative): casi negativi correttamente classificati
  - TP (True Positive): casi positivi correttamente classificati

- FP (False Positive): casi negativi erroneamente classificati come positivi
- FN (False Negative): casi positivi erroneamente classificati come negativi

Metriche derivate dalla matrice di confusione:

- **Accuracy**: (TP+TN)/(TP+TN+FP+FN)
- **Precision**: TP/(TP+FP)
- **Recall (Sensitivity)**: TP/(TP+FN)
- **False Positive Rate**: FP/(TN+FP)
- **F1 Score**: 2 *(Precision* Recall) / (Precision + Recall)

5.10.4 Curva ROC e AUC

La curva ROC (Receiver Operating Characteristic) per un classificatore viene creata registrando il valore di True Positive Rate rispetto a False Positive Rate al variare di un parametro (spesso la soglia utilizzata per discriminare tra due classi).

L'area sotto la curva (Area Under the Curve; AUC) varia tra 0 e 1 e rappresenta la performance del classificatore (AUC=1 significa che le predizioni sono corrette al 100%).

L'AUC è invariante alla soglia, quindi misura la qualità delle predizioni del modello indipendentemente dalla soglia di classificazione scelta.

6. DEEP LEARNING SUPERVISIONATO

6.1 Introduzione al Deep Learning

Il deep learning supervisionato si riferisce all'addestramento di reti neurali profonde (con molti strati nascosti) utilizzando dati etichettati.

Nel deep learning la rete neurale ha un numero di strati nascosti > 1 (che può essere nell'ordine delle decine) formando quindi una rete profonda (deep neural network).

6.1.1 Fattori di Successo del Deep Learning

Due fattori hanno grande importanza per il successo del deep learning:

- **Big data**: accesso a grandi quantità di dati etichettati
- **GPU computing**: enorme potenza di calcolo parallelo su schede di processori grafici

6.1.2 Caratteristiche Distintive

Caratteristica distintiva del deep learning rispetto ad altre tecniche di machine learning è che l'apprendimento può avvenire direttamente sui dati grezzi, senza pre-processing / estrazione di features. Queste ultime vengono "scoperte" dalla rete durante l'apprendimento.

6.2 Elaborazione Gerarchica

6.2.1 Un Principio Generale di Organizzazione Neurale

L'elaborazione gerarchica è un principio generale di organizzazione neurale, documentato da studi come quelli di Felleman & Van Essen (1991) e Riesenhuber & Poggio (1999), dove la codifica dell'informazione diventa più complessa ed astratta passando a livelli più elevati (profondi) della gerarchia.

6.2.2 Ottenere Separabilità Lineare attraverso Proiezioni Non Lineari

Le reti neurali profonde riescono a trasformare problemi difficili (non linearmente separabili nello spazio degli input) in problemi più semplici (linearmente separabili nello spazio delle features) attraverso successive trasformazioni non lineari.

Questo processo può essere visto come un "districamento" (disentanglement) delle "varietà" (manifolds) che rappresentano le diverse classi nello spazio degli input.

6.3 Il Problema del "Vanishing Gradient"

Se aggiungiamo molti strati nascosti, il segnale di errore deve passare molti livelli di retro-propagazione prima di giungere alle connessioni più vicine all'input.

Questo fenomeno è particolarmente problematico con la funzione di attivazione sigmoide, che tende a saturare quando la somma pesata degli input è troppo grande o troppo piccola. La derivata tende a zero nella zona di saturazione, quindi il gradiente svanisce nel giro di pochi passi di retro-propagazione.

6.4 Recenti Sviluppi che hanno Consentito la Rivoluzione del Deep Learning

6.4.1 Miglioramento degli Algoritmi

- **Inizializzazione dei pesi più "intelligente"**: invece che totalmente random, si utilizzano tecniche come l'inizializzazione di Xavier o He
- **Learning rate adattivo**: in base al gradiente (ADAM, AdaGrad, RMSProp...)
- **Utilizzo di reti di grandi dimensioni** combinate con efficaci regolarizzatori (sparsità + decadimento dei pesi + dropout)
- **Ottimizzatori del 2° ordine**: derivata seconda per stimare la curvatura del gradiente
- **Funzione di attivazione che non satura il gradiente**: Rectified Linear Unit (ReLU)

6.4.2 Big Data e Hardware di Calcolo Parallelo

- **Disponibilità di enormi training set digitali annotati** (Big Data): esempi come ImageNet con 1.2 milioni di immagini in 1000 categorie
- **Aumento delle prestazioni di calcolo**: hardware di calcolo parallelo, GPU
- **Per applicazioni di computer vision**: utilizzo di architetture convoluzionali

6.5 Hardware per Calcolo Parallelo

- Architetture di calcolo parallele consentono di sfruttare il parallelismo inerente delle reti neurali [Parallel Distributed Processing]
- GPU (Graphic Processing Unit) --> schede video inizialmente sviluppate per il gaming, ora utilizzate per addestrare reti neurali
- Con GPU è possibile addestrare reti neurali gerarchiche con migliaia / milioni di neuroni e connessioni sinaptiche

6.6 Reti Neurali Convoluzionali (CNN)

6.6.1 Definizione e Struttura

La CNN è una rete profonda che include almeno uno strato convoluzionale (convolution layer), in cui i neuroni nascosti non sono interamente connessi con lo strato precedente ma hanno campi recettivi locali.

Lo strato convoluzionale è di solito seguito da uno strato di pooling (pooling layer) per ridurre la dimensionalità ed enfatizzare le caratteristiche più salienti.

Nella parte più profonda della rete è inserito almeno uno strato nascosto "standard", ovvero interamente connesso (fully connected) con quello precedente.

L'ultimo strato nascosto (fully connected) attiva lo strato di output (categorie).

6.6.2 Lo Strato Convoluzionale

- Ogni neurone nascosto dello strato convoluzionale ha un campo recettivo locale, che codifica una feature specifica (chiamata anche kernel o filtro)
- Il numero di neuroni definisce quante features verranno rappresentate in ciascun livello
- A differenza delle reti multi-strato standard, ciascun filtro è applicato all'intera immagine attraverso un'operazione di convoluzione
- È come avere più copie dello stesso neurone: utilizziamo lo stesso insieme di pesi per processare diverse porzioni dell'immagine in input (condivisione dei parametri)

6.6.3 L'Operazione di Convoluzione

La convoluzione è un'operazione matematica che applica un filtro ad un'immagine per estrarne caratteristiche specifiche:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

dove:

- I è l'immagine di input
- K è il kernel (filtro)
- S è l'output della convoluzione

Diversi filtri generano in output una mappa diversa a partire dalla stessa immagine, enfatizzandone una particolare caratteristica (feature map).

6.6.4 Iperparametri delle Reti Convoluzionali

- **Numero di neuroni nascosti**: specifica quanti filtri usare in ciascun layer
- **Dimensione kernel**: definisce il campo recettivo del filtro
- **Stride**: dimensione del passo di spostamento del filtro (overlap)
- **Padding**: per mantenere invariata la dimensione dell'immagine, si imposta un bordo aggiuntivo con valori costanti (solitamente a zero)

6.6.5 Lo Strato di Pooling

Lo strato di pooling viene inserito dopo uno strato convoluzionale per ridurre la dimensione dell'immagine (operazione di max pooling o average pooling).

Questo riduce il numero di parametri, controllando l'overfitting, e promuove l'invarianza (es. alla traslazione).

6.7 Architetture CNN Famose

6.7.1 LeNet-5

- LeCun et al. (1998), Proceedings of the IEEE
- 7 strati (5 convoluzionali, 3 fully-connected)
- Immagini in bianco e nero 32x32 pixels (MNIST dataset)
- Prima CNN di successo, utilizzata per il riconoscimento di cifre scritte a mano

6.7.2 AlexNet

- Krizhevsky et al. (2012), NeurIPS
- 8 strati (5 convoluzionali, 3 fully-connected)
- Raggiunse lo stato dell'arte (a quel tempo) operando su immagini a dimensione reale
- Consentì un incremento dell'accuratezza del 10% nei sistemi di computer vision
- Prima rete neurale "su larga scala", addestrata per 6 giorni su ImageNet utilizzando 2 GPU

6.7.3 GoogLeNet (Inception)

- Szegedy et al. (2014), CVPR
- Architettura "inception" con 22 strati
- Raggiunge prestazione di classificazione comparabile a quella umana in ImageNet
- Rete neurale con milioni di connessioni, addestrata per diverse settimane nei supercomputer di Google

6.7.4 ResNet

- He et al. (2015)
- Usa la tecnica di "mapping residuo" per risolvere il problema della saturazione e degradazione della performance quando il numero di strati aumenta
- Le connessioni dirette ("shortcut" o "residual" connections) eseguono un mapping che è la funzione identità
- La funzione che descrive il mapping input-output, $y = f(x)$, può essere riscritta come $y = f(x) + x$
- Lo schema di ResNet si può applicare a strati di diverso tipo (convoluzionali e non)
- Versioni con 50, 101, e persino 152 strati

6.8 Successi e Applicazioni del Deep Learning Supervisionato

- Riconoscimento di immagini: prestazioni comparabili o superiori a quelle umane
- Diagnosi medica: identificazione di patologie da immagini mediche
- Guida autonoma: riconoscimento di oggetti, pedoni, segnali stradali
- Riconoscimento del linguaggio parlato: trascrizione del parlato in testo
- Riconoscimento facciale: identificazione di persone da immagini del volto

6.9 Limitazioni e Problemi

6.9.1 Vulnerabilità alla Distorsione

Le CNN possono essere sensibili a distorsioni e manipolazioni dell'input:

- Rumore nell'immagine
- Cambiamenti di posizione, scala, rotazione (se non sufficientemente rappresentati nel training set)
- Condizioni di illuminazione

6.9.2 Attacchi Avversari

Possiamo indurre errori di classificazione in una rete neurale profonda facendo modifiche ad hoc all'immagine di input. Questi esempi avversari (adversarial examples) sembrano del tutto innocui all'occhio umano e rappresentano una seria minaccia in contesti sensibili (cyber-sicurezza).

- La modifica può essere una perturbazione (rumore) del valore dei pixel
- L'immagine non è distinguibile dall'originale all'occhio umano
- Una specifica perturbazione può avere lo stesso effetto su una diversa rete profonda

7. RETI RICORRENTI E APPRENDIMENTO SELF-SUPERVISED

7.1 La Dimensione Temporale nell'Apprendimento

L'informazione in input può arrivare al sistema neurale in modo sequenziale e quindi può possedere una struttura temporale. Anche l'output può avere una natura sequenziale.

Esempi di applicazioni con struttura temporale:

- Riconoscimento del parlato (da spettrogramma audio a sequenza di fonemi e/o parole)
- Riconoscimento di azioni
- Predizione della dinamica in oggetti in movimento
- Generazione di testo o di musica
- Predizione di serie storiche: item(n+1) = f(item(n))

7.2 Come Apprendere Compiti Sequenziali?

In una classica rete multistrato l'output O(t) al tempo t dipende solo dall'input corrente I(t). Quindi non può apprendere dipendenze temporali.

7.2.1 Approccio con Finestra Temporale

Una possibile soluzione: trasformare il tempo in spazio, presentando come input alcuni elementi della sequenza S in una "finestra temporale" W_T(t) che si sposta sopra S: W_T(t) = [S_t, S_t+1, ..., S_t+T]

La rete ha quindi a disposizione nell'input il "contesto" (es. elementi precedenti) necessario per l'apprendimento.

Problemi di questo approccio:

1. Dimensionalità dell'input: i neuroni di input vengono replicati per ogni elemento rappresentato nella "finestra"
2. Contesto limitato dalla grandezza della finestra

7.2.2 Esempio: NetTalk

NetTalk (Sejnowski & Rosenberg, 1987) è un esempio di modello basato su finestra temporale:

- Compito: leggere le parole inglesi
- Rete neurale: MLP
- Input: una lettera e le tre lettere vicine a sinistra e destra (7 lettere totali)
- 80 unità nascoste
- Output: il fonema corrispondente alla lettera centrale

7.3 Reti Parzialmente Ricorrenti (Simple Recurrent Networks – SRN)

7.3.1 Principio di Funzionamento

Un modo semplice di elaborare strutture temporali consiste nell'aggiungere connessioni ricorrenti che forniscano un input ricorrente proveniente dello stesso strato o da strati superiori.

In questo modo, le risposte delle reti dipendono da:

1. Input "classico" feed-forward dagli strati inferiori
2. Input da stati precedenti

Lo stato S(t) delle reti ricorrenti dipende non solo dallo stato precedente S(t-1), ma anche da tutti gli stati passati S(1)...S(t-1).

7.3.2 Addestrare Reti Ricorrenti

Le reti parzialmente ricorrenti si possono addestrare con l'algoritmo back-propagation, perché la struttura temporale può essere "srotolata" trasformandola in una struttura spaziale.

Ad esempio, la sequenza di input S=[x_1 x_2] viene trasformata in una serie di passi che possono essere appresi da una rete statica.

Back-propagation through time (BPTT): Aggiorniamo i pesi rispetto all'intera sequenza aggregando (sommando) i gradienti calcolati per ogni passo nel tempo (livello nella rete srotolata).

7.3.3 Le Reti di Elman (SRN)

Le reti di Elman (Elman, 1990) sono un tipo di rete ricorrente semplice:

- Neuroni di output: $y_i^t = sigm(\sum_j w_{ij}^t h_j^t)$
- Neuroni nascosti: $h_i^t = sigm(\sum_j w_{ij}^t x_j^t + \sum_k w_{ik}^t c_k^t)$

- Neuroni contesto: c_i^t = h_i^(t-1) è la copia dello stato dei neuroni nascosti h_i^(t-1) (al passo precedente)

7.4 Apprendimento da Sequenze: Self-Supervised Learning

7.4.1 Definizione e Principio

L'apprendimento self-supervised è un tipo di apprendimento dove il sistema genera automaticamente le etichette dei dati a partire dai dati stessi, senza bisogno di supervisione esterna.

Nel contesto delle sequenze, il compito tipico è:

- Training set: sequenze di elementi (es. lettere, parole, frame video)
- Compito: predire il prossimo elemento nella sequenza

Questo tipo di apprendimento viene chiamato self-supervised perché input e target hanno la stessa natura (sono entrambi elementi della sequenza).

7.4.2 Scoprire la Struttura Temporale

Elman (1990) ha dimostrato che le reti ricorrenti possono scoprire strutture linguistiche:

- L'errore di predizione è elevato all'inizio delle parole (dove la prevedibilità è minore)
- Le rappresentazioni interne (attivazioni delle unità nascoste) corrispondenti ad ogni parola mostrano una struttura
- Analizzando queste rappresentazioni con clustering gerarchico, si scopre che la rete raggruppa le parole in base al loro utilizzo, ovvero al contesto circostante, scopendo diverse categorie lessicali

7.5 Long-Short Term Memory (LSTM) Networks

7.5.1 Il Problema della Memoria a Lungo Termine

Le SRN hanno una memoria "a breve termine" (short-term memory): non riescono ad apprendere dipendenze temporali lontane nella sequenza di input.

7.5.2 La Soluzione LSTM

Questo problema viene risolto nelle reti LSTM, in cui lo strato nascosto è formato da unità LSTM che hanno una struttura più complessa rispetto ai tipici neuroni nascosti.

Le unità LSTM, proposte da Hochreiter & Schmidhuber (1997), operano attraverso porte ("gates") che definiscono (attraverso l'apprendimento) se l'informazione vada mantenuta nella memoria temporanea e quanto avanti nel tempo dovrebbe essere propagata:

- **Input gate**: lascia entrare l'input nella cella di memoria
- **Output gate**: lascia uscire il valore corrente della cella di memoria
- **Forget gate**: resetta il valore corrente nella cella di memoria

NB: ogni gate ha il suo insieme di pesi sinaptici!

7.5.3 Architettura di una Unità LSTM

L'unità LSTM è composta da:

- Una cella di memoria (cell state) che mantiene l'informazione nel tempo
- Tre porte (gates) che controllano il flusso di informazione:
  - Forget gate: decide quali informazioni scartare
  - Input gate: decide quali informazioni nuove aggiungere
  - Output gate: decide quale output produrre

Matematicamente, l'operazione di una unità LSTM al tempo t può essere descritta come:

Forget gate: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$

Input gate: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

Candidato per la cella: $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$

Aggiornamento della cella: $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

Output gate: $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$

Output: $h_t = o_t * \tanh(C_t)$

Dove σ è la funzione sigmoide, $*$ è moltiplicazione elemento per elemento, e $[h_{t-1}, x_t]$ è la concatenazione dell'output precedente e dell'input corrente.

7.6 Deep Recurrent Neural Networks (RNN) per Sequenze

7.6.1 Architettura e Funzionamento

Le Deep RNN sono reti profonde con strati nascosti multipli di unità LSTM:

- L'input è un elemento della sequenza (es., un frame video)
- L'output è il prossimo elemento della sequenza (es., il prossimo frame)

7.6.2 Tipi di Predizione

- **Predizione one-step**: la rete viene testata sulla predizione del prossimo elemento, come durante l'apprendimento (es, un frame avanti)

- **Predizione multi-step**: la predizione del prossimo elemento è utilizzata come input per ottenere la predizione del successivo (es., due frames avanti)

7.6.3 Esempio: Generazione di Testo

Le Deep RNN possono essere addestrate su corpus linguistici di larga scala (Wikipedia, New York Times, articoli scientifici) per generare testo a livello di carattere (ogni elemento nella sequenza è un singolo carattere o simbolo).

Esempi di testo generato (Sutskever et al., 2011):

- Da RNN addestrata su Wikipedia iniziando con "The meaning of life is...": "The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger. In the show's agreement unanimously resurfaced. [...]"
- Da RNN addestrata su articoli di machine learning iniziando con "Recurrent...": "Recurrent network with the Stiefel information for logistic regression methods Along with either of the algorithms previously (two or more skewprecision) is more similar to the model with the same average mismatched graph. [...]"

7.7 Word Embeddings

7.7.1 Definizione e Concetto

La modellizzazione di sequenze per l'elaborazione del linguaggio naturale (NLP) è più efficiente al livello della parola. Questo richiede di codificare le singole parole con vettori di lunghezza fissa usando algoritmi di embedding.

Word embedding è un termine usato per la rappresentazione di parole in forma di un vettore numerico che codifica il significato della parola.

7.7.2 Proprietà degli Embeddings

- **Conservazione di relazioni semantiche**: parole vicine nello spazio vettoriale (es., distanza angolare tra vettori) hanno un significato simile
- **Composizionalità**: operazioni lineari sui vettori possono dare risultati coerenti
  - Esempio: vec("Madrid") – vec("Spain) + vec("France") è più vicino a vec("Paris") che a qualunque altro vettore

7.7.3 Algoritmi di Word Embedding

Esistono vari algoritmi per creare embeddings di parole:

- **Word2Vec** (Mikolov et al., 2013):
  - CBOW (Continuous Bag of Words): predice una parola dato il suo contesto

    - Skip-gram: predice il contesto di una parola data la parola
- **GloVe** (Global Vectors, Pennington et al., 2014):
    - Combina statistiche globali e apprendimento locale per creare vettori
- **FastText** (Bojanowski et al., 2017):
    - Estensione di Word2Vec che considera anche i subword tokens (n-grammi di caratteri)

7.8 Transformers

7.8.1 Definizione e Principio

Transformer è un'architettura di rete neurale che si basa su meccanismi di self-attention per trasformare una sequenza di elementi (tokens) in input in una sequenza di elementi in output, senza utilizzare convoluzioni o connessioni ricorrenti (Vaswani et al., 2017, "Attention is all you need").

7.8.2 Architettura Generale

L'architettura generale del Transformer prevede due blocchi principali, spesso utilizzati in modo indipendente:

Encoder:

- Costruisce una rappresentazione interna della sequenza
- Utilizza come contesto per ogni token sia gli elementi precedenti (a sinistra) che i successivi (a destra)
- Esempio: BERT (Bidirectional Encoder Representations from Transformers) di Google
    - Addestrato a predire una parola "mascherata" utilizzando come contesto l'intera frase (masked language modeling)
    - Es: "Mercoledì prossimo la lezione sarà in XXX informatica." → "aula"

Decoder:

- Genera una sequenza di tokens utilizzando solo i tokens precedenti (a sinistra) come contesto per condizionare la generazione
- Esempio: GPT (Generative Pretrained Transformer) di OpenAI
    - Addestrato in modo autoregressivo a predire la parola seguente utilizzando come contesto solo le parole precedenti (causal language modeling)
    - Es: "Mercoledì prossimo la lezione sarà in" → "aula"

7.8.3 Self-Attention

Self-attention si riferisce al fuoco dell'attenzione, per ogni token, rispetto a tutti gli altri tokens di input.

Il meccanismo di self-attention permette a ogni posizione nella sequenza di prestare attenzione a tutte le posizioni nella sequenza stessa, assegnando pesi diversi a diverse posizioni.

Matematicamente, l'attenzione può essere descritta come:

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

Dove:

- Q (query): una matrice di vettori di query
- K (key): una matrice di vettori di chiave
- V (value): una matrice di vettori di valore
- d_k: la dimensione dei vettori di chiave

7.8.4 Componenti dei Transformers

- **Positional Encoding**: informazione sulla posizione relativa o assoluta dell'elemento nella sequenza
- **Multi-head Attention**: permette di catturare diverse relazioni tra tokens in parallelo
- **Feed-forward Networks**: elaborano le rappresentazioni in ogni posizione in modo indipendente
- **Layer Normalization**: normalizza le attivazioni per stabilizzare l'apprendimento
- **Residual Connections**: permettono il flusso di informazioni attraverso strati profondi

7.9 Large Language Models (LLM)

7.9.1 Definizione e Caratteristiche

I Large Language Models (LLM) sono modelli basati sull'architettura Transformer e addestrati su enormi quantità di testo scaricato da Internet (Wikipedia, giornali, libri, etc).

Caratteristiche principali:

- Aumentare la scala del Transformer (numero di parametri = numero di connessioni) migliora la performance
- Il modello linguistico viene addestrato con apprendimento self-supervised (predizione della parola seguente)
- Successivo "raffinamento" (fine-tuning) su compiti specifici con apprendimento supervisionato

7.9.2 Allineamento degli LLM

Per migliorare l'accuratezza e ridurre le risposte inappropriate (inclusi bias e linguaggio "tossico") il LLM deve essere "allineato" usando:

- Feedback di utenti umani
- Reinforcement Learning from Human Feedback (RLHF): ulteriore fine-tuning basato su apprendimento per rinforzo

7.9.3 Generazione di Testo in GPT

Un LLM viene addestrato con self-supervised learning a predire il prossimo token data l'attuale sequenza di token di testo come input. I tokens possono essere parole, parti di parole (es. suffissi), punteggiatura.

La generazione è sequenziale:

- Un token ad ogni passo
- Ottenuto aggiungendo in modo iterativo l'output al contesto
- Ogni token ha una probabilità
- Si possono utilizzare specifici metodi di decoding per selezionare il token, rendendo l'output più variabile ed interessante

7.9.4 GPT-3 e Successivi

GPT-3 ha rappresentato un salto qualitativo nei LLM:

- 175 miliardi di parametri
- Training su 570 milioni di Gigabytes di testo (un calcolo grossolano è che un essere umano dovrebbe vivere 1000 anni per essere esposto a questa quantità di linguaggio)
- Primo LLM reso disponibile al pubblico in forma di chatbot (ChatGPT)
- Few-shot learning: proprietà emergente di eseguire compiti senza addestramento specifico mostrando solo alcuni esempi come parte del prompt

I successivi modelli (GPT-4, Claude, etc.) hanno migliorato ulteriormente le capacità generative e di comprensione.

8. APPRENDIMENTO NON SUPERVISIONATO

8.1 Introduzione all'Apprendimento Non Supervisionato

L'apprendimento non supervisionato è un paradigma di machine learning in cui l'algoritmo impara a trovare pattern o strutture nei dati senza utilizzare etichette di supervisione.

L'obiettivo principale è di scoprire struttura latente (regolarità statistiche) nei dati.

8.1.1 Vantaggi dell'Apprendimento Non Supervisionato

- L'apprendimento non richiede nessuna etichetta negli esempi. Ciò significa che il sistema può sfruttare l'enorme quantità di informazione "grezza" presente nell'ambiente

- Una volta appreso un buon modello interno dell'ambiente (processo di estrazione di features o di apprendimento di rappresentazioni), esso può essere utilizzato anche per apprendere più facilmente task supervisionati (transfer learning)
- Sembra che gli animali (uomo incluso) sfruttino massivamente questa modalità di apprendimento durante lo sviluppo

8.1.2 Svantaggi dell'Apprendimento Non Supervisionato

- Spesso non è chiaro quali features dell'ambiente saranno poi utili per risolvere un determinato compito, o cosa costituisca una "buona" rappresentazione
- L'apprendimento non supervisionato richiede molte risorse computazionali. A volte sarebbe più conveniente avere un esperto (teacher) che aiuta a risolvere un compito
- Dalla semplice osservazione dell'ambiente non possiamo inferire relazioni causali

8.1.3 Principali Algoritmi di Apprendimento Non Supervisionato

- Riduzione lineare della dimensionalità (PCA)
- Riduzione non-lineare della dimensionalità (Autoencoders)
- Apprendimento associativo (reti di Hopfield)
- Modelli generativi
  - Macchine di Boltzmann e Deep Belief Networks
  - Generative Adversarial Networks, Transformers
  - Normalizing flows / Diffusion models
- Apprendimento competitivo (self-organizing maps)
- Altri algoritmi di clustering (es: k-means, clustering gerarchico)
- Modelli di misture (es: mistura di Gaussiane)
- Hidden Markov Models
- Analisi fattoriale

8.2 L'Importanza di Scoprire Buone Rappresentazioni

8.2.1 Approcci Alternativi per il Riconoscimento

1. **Apprendimento diretto dai dati grezzi**:
   - Si apprende un mapping supervisionato direttamente dai dati grezzi alle classi
   - Approccio potenzialmente difficile quando i dati sono complessi
2. **Estrazione di features e poi mapping**:
   - Prima si estraggono una serie di features descrittive (caratteristiche)
   - Poi si esegue il mapping basandosi su queste rappresentazioni
   - Questo approccio può rendere il compito di classificazione molto più semplice

8.2.2 Interpretazione Geometrica: "Districare Manifolds"

Nelle reti neurali profonde, i diversi strati di elaborazione possono essere visti come trasformazioni successive che "districano" (disentangle) le varietà (manifolds) di dati che nei dati grezzi sono altamente intrecciate e non separabili linearmente.

Man mano che si sale nella gerarchia della rete, le rappresentazioni diventano sempre più astratte e separabili linearmente.

8.3 Approcci per Apprendere Rappresentazioni

8.3.1 Clustering vs. Estrazione di Features

Idea di base: vogliamo rimuovere l'informazione non necessaria

Clustering:

- Riduce il numero di esempi raggruppandoli insieme creando "prototipi"
- Es: se Esempio 1 ed Esempio n sono estremamente simili, li raggruppiamo

Feature extraction:

- Riduce il numero di descrittori, considerando solo i più informativi o creandone di più astratti
- Es: una feature che è sempre costante può essere eliminata
- Es: "Dimensione" può astrarre "Area" e "Perimetro"

Il risultato in entrambi i casi è una riduzione della dimensionalità dei dati.

8.4 Riduzione Lineare della Dimensionalità: PCA

8.4.1 Definizione e Concetto

La Principal Component Analysis (PCA) è una tecnica statistica che cerca di trovare la direzione di massima variabilità in un certo insieme di dati.

Più precisamente, lo scopo della PCA è di scoprire un insieme di variabili linearmente scorrelate (le "componenti principali") che spieghino la maggior parte della varianza osservata nella distribuzione.

Spesso PCA cattura la maggior parte della varianza in poche dimensioni (< 5).

8.4.2 Comprimere Dati tramite PCA

La scomposizione dei dati data dalla PCA può essere usata per mappare i pattern di input in un nuovo sistema di coordinate a bassa dimensionalità.

Possiamo rappresentare il nostro dataset di input con la matrice X, e decomporre questa matrice come prodotto di due matrici:

- Matrice dei "loadings" (pesi): definisce le direzioni delle componenti principali
- Matrice degli "scores" (punteggi): definisce le coordinate dei dati originali nel nuovo sistema

Meno componenti consideriamo, peggiore sarà la ricostruzione, ma i dati saranno più compressi.

8.4.3 Procedura per Calcolare la PCA

1. Standardizzare i dati (sottrarre la media e dividere per la deviazione standard)
2. Calcolare la matrice di covarianza
3. Calcolare gli autovalori e gli autovettori della matrice di covarianza
4. Ordinare gli autovettori in base ai corrispondenti autovalori (dal più grande al più piccolo)
5. Scegliere i primi k autovettori (componenti principali)
6. Trasformare i dati originali nello spazio delle componenti principali

8.5 Autoencoders

8.5.1 Definizione e Struttura

Un autoencoder è una rete neurale che impara a codificare i dati di input in una rappresentazione di dimensione inferiore (encoding) e poi a ricostruire l'input originale da questa rappresentazione (decoding).

Possiamo costruire un autoencoder che riduce la dimensionalità dei dati semplicemente chiedendo che "ricostruisca" in output il pattern presentato in input.

La funzione di errore è rappresentata dall'errore di ricostruzione (es: scarto quadratico medio):

$$E = \frac{1}{2} \sum_i (t_i - y_i)^2$$

8.5.2 Strato Nascosto e Compressione

Se "forziamo" la rete neurale a codificare i dati di input utilizzando un numero ridotto di neuroni nascosti, imparerà a comprimere la distribuzione dei dati estraendone le features più rilevanti.

- Utilizzando neuroni nascosti con funzione di attivazione lineare, il risultato sarà molto simile a quello ottenuto con la PCA
- Utilizzando neuroni nascosti con funzione di attivazione non lineare (es: sigmoide o ReLU), effettueremo invece una riduzione non-lineare della dimensionalità

8.5.3 Denoising Autoencoders

L'estrazione di features può essere resa più robusta introducendo rumore nei dati durante l'addestramento:

- Il rumore funge da regolarizzatore (in modo analogo a weight decay o dropout)
- La rete deve imparare a ricostruire il dato originale a partire da una versione rumorosa
- Questo costringe la rete a catturare le caratteristiche più importanti e robuste dei dati

8.5.4 Analisi dello Spazio Latente

Utilizzando solamente due neuroni nascosti possiamo persino comprimere i dati in input in un piano bidimensionale, che permette di visualizzare lo spazio latente.

In questo spazio, punti vicini corrispondono a dati con caratteristiche simili nel dominio originale.

8.5.5 Autoencoders Variazionali (VAE)

Negli ultimi anni sono stati introdotti modelli di autoencoders più performanti, che introducono ulteriori regolarizzatori nello spazio latente:

- Il VAE forza la distribuzione dello spazio latente ad approssimare una distribuzione normale multivariata
- Ciò consente di generare nuovi dati campionando punti dallo spazio latente
- La funzione di perdita include un termine di divergenza KL che misura quanto la distribuzione latente si discosta da una normale

8.6 Apprendimento Associativo e Reti di Hopfield

8.6.1 Dalla Regola di Hebb ai Modelli Associativi

La regola di Hebb scopre correlazioni nei dati: "Neurons wire together if they fire together".

Possiamo usare la regola di Hebb per apprendere i pesi sinaptici di reti più complesse, permettendo di memorizzare interi pattern.

8.6.2 Ispirazione dalla Meccanica Statistica

La meccanica statistica mette in relazione proprietà microscopiche di atomi e molecole con proprietà macroscopiche della materia:

- Modelli di Ising: componenti elementari (atomi) interagiscono con i loro vicini in un reticolo bi-dimensionale
- Ciascun atomo può trovarsi in due possibili stati discreti (spin)
- Il sistema diventa ferromagnetico quando tutti gli spin sono allineati

- La dinamica del sistema è governata da una funzione di energia
- Il sistema esplora varie configurazioni, e si assesta in quelle più probabili (ovvero, quelle con energia minima)

La "frustrazione geometrica" fa sì che il sistema non raggiunga uno stato uniforme, ma si assesti in configurazioni eterogenee dove l'energia è minimizzata localmente.

8.6.3 Reti di Hopfield

Le reti di Hopfield sostituiscono "atomi" con "neuroni" e permettono alle interazioni locali di modificarsi attraverso apprendimento Hebbiano.

Caratteristiche delle reti di Hopfield:

- Possono essere usate per memorizzare e recuperare pattern
- La memorizzazione avviene cambiando gradualmente la forza delle connessioni
- Il recupero avviene in modo dinamico, aggiornando iterativamente lo stato dei neuroni finché non si raggiunge uno stato stabile (attrattore)
- In analogia con la meccanica statistica, si usa una funzione di energia per specificare quali stati della rete sono più probabili

Architettura:

- La rete è ricorrente (tutte le connessioni sono bidirezionali) con topologia completamente connessa
- Non ci sono auto-connessioni
- Le reti di Hopfield possono essere interpretate come modelli grafici non direzionati (pesi simmetrici)
- Tutti i neuroni sono "visibili", ovvero ciascuno corrisponde ad una variabile in input

8.6.4 Energia ed Attrattori

L'energia E di una certa configurazione delle attivazioni dei neuroni è data da:

$$E = -\frac{1}{2}\sum_{i,j} w_{ij} x_i x_j$$

Gli attrattori della rete corrispondono ai punti di minimo locale della funzione di energia, e rappresentano pattern stabili di attività che codificano una "memoria" (ovvero, un determinato pattern di input).

8.6.5 Minimizzare l'Energia

Ci sono due meccanismi che consentono di minimizzare l'energia:

- **Inferenza**: aggiorniamo le attivazioni dei neuroni per farle diventare più simili a quelle osservate durante il training (pattern memorizzati come attrattori)
- **Apprendimento**: aggiorniamo i pesi delle connessioni per creare gli attrattori in corrispondenza dei pattern di training

8.6.6 Inferenza nelle Reti di Hopfield

I neuroni hanno stati binari (-1 o 1) e la loro attivazione è calcolata utilizzando la regola:

$$x_i = \begin{cases} +1 & \text{se } \sum_j w_{ij} x_j \geq \theta_i \\ -1 & \text{altrimenti} \end{cases}$$

Hopfield ha dimostrato che, se i pesi hanno valori appropriati, le attivazioni della rete convergeranno sempre verso uno stato stabile (attrattore), dove le attivazioni non cambiano più (la rete raggiunge l'equilibrio).

8.6.7 Apprendimento nelle Reti di Hopfield

I pesi della rete vengono impostati usando apprendimento Hebbiano:

- Ogni pattern di training viene presentato iterativamente alla rete "vincolandolo" nei neuroni
- Le connessioni fra i neuroni vengono modificate in base alla regola:

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

L'apprendimento "scava" il bacino degli attrattori, creando minimi locali nella funzione di energia corrispondenti ai pattern di training.

8.6.8 Limitazioni e Soluzioni

Attrattori spuri: Se incrementiamo il numero di pattern (attrattori) da memorizzare, la rete svilupperà anche attrattori spuri, che non corrispondono a minimi locali ma rappresentano stati di equilibrio.

Dinamica stocastica: Per evitare di rimanere intrappolati in cattivi minimi locali, possiamo sostituire la funzione di attivazione deterministica con una funzione stocastica:

$$P(x_i = 1) = \frac{1}{1 + e^{-\frac{1}{T} \sum_j w_{ij} x_j}}$$

Il parametro T rappresenta la "Temperatura" del sistema, in analogia con i materiali fisici.

Simulated Annealing: Per raggiungere il miglior minimo di energia, cominciamo con una temperatura alta nel sistema, e poi gradualmente lo "raffreddiamo", riducendo progressivamente la temperatura finché il sistema diventa deterministico.

8.7 Modelli Generativi con Variabili Latenti

8.7.1 Limitazioni delle Reti di Hopfield

Nelle reti di Hopfield tutti i neuroni sono visibili, perciò possiamo scoprire solo correlazioni "pairwise" (a coppie).

8.7.2 Il Cervello "Bayesiano"

La percezione e la cognizione possono essere caratterizzate come processi inconsci di inferenza statistica:

- **Inferenza**: data una certa evidenza (variabili visibili, osservate), calcolare la distribuzione di probabilità delle variabili nascoste (ipotesi, non osservate)
- **Apprendimento**: trovare i parametri (pesi delle connessioni) del modello generativo che meglio descrive (riproduce) la distribuzione dei dati osservati

Il teorema di Bayes ci fornisce il formalismo per l'inferenza:

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)}$$

8.7.3 Reti Neurali Generative

Invece di apprendere un mapping input-output, un modello generativo cerca di scoprire la struttura latente dei dati in input creando un modello interno (probabilistico) dell'ambiente, che potrebbe aver generato tali dati.

Le cause latenti di un segnale sensoriale costituiscono la sua rappresentazione interna. In ottica Bayesiana, rappresentano le ipotesi sui dati, che vengono aggiornate ogni volta che osserviamo nuova evidenza.

9. DEEP LEARNING NON SUPERVISIONATO

9.1 Restricted Boltzmann Machine (RBM)

9.1.1 Dalle Reti di Hopfield alle Macchine di Boltzmann

Le macchine di Boltzmann sono una variante stocastica delle reti di Hopfield che sfruttano unità nascoste per estrarre correlazioni di ordine superiore dai dati.

Usando neuroni nascosti, le macchine di Boltzmann ottengono una capacità di memorizzazione superiore alle reti di Hopfield. I neuroni nascosti vengono usati per comprimere l'informazione, dato che estraggono struttura latente (features) dalla distribuzione dei dati.

Le macchine di Boltzmann complete sono computazionalmente molto dispendiose!

9.1.2 Restricted Boltzmann Machine (RBM)

La RBM è un grafo bipartito, completamente connesso (no connessioni intra-layer):

- I neuroni nascosti ("ipotesi") diventano condizionalmente indipendenti
- La complessità computazionale si riduce enormemente

La funzione di energia dipende dai valori delle connessioni sinaptiche (pesi W), che costituiscono i parametri del modello generativo:

$$E(v,h) = -\sum_i \sum_j v_i h_j w_{ij} - \sum_i a_i v_i - \sum_j b_j h_j$$

dove:

- v è il vettore delle unità visibili
- h è il vettore delle unità nascoste
- w_{ij} è il peso tra l'unità visibile i e l'unità nascosta j
- a_i e b_j sono i bias delle unità visibili e nascoste

9.1.3 Divergenza Contrastiva

L'idea di base dell'algoritmo di Divergenza Contrastiva è di minimizzare la discrepanza fra la distribuzione empirica dei dati (pattern nel training set) e la distribuzione generata dal modello (pattern creati dalla rete quando lo strato visibile non è vincolato ai dati).

Intuitivamente, vogliamo fare in modo che le "distorsioni" dei dati prodotte dal modello assomiglino il più possibile ai dati "veri" (training set).

9.1.4 Algoritmo della Divergenza Contrastiva (CD-1)

Per ciascun pattern di training:

Fase positiva:

1. Il pattern è presentato alla rete ("vincolato" [clamped] nei neuroni visibili v)
2. L'attivazione dei neuroni nascosti h viene calcolata in un singolo step usando la funzione di attivazione stocastica
3. Calcoliamo le correlazioni $< v^+ h^+ >$ fra le unità visibili e le unità nascoste

Fase negativa:

1. Partendo dalle attivazioni delle unità nascoste calcolate durante la fase positiva, si generano le attivazioni nel layer visibile usando la funzione di attivazione stocastica
2. Partendo da queste nuove attivazioni visibili, ricalcoliamo le attivazioni nascoste
3. Calcoliamo le correlazioni $< v^- h^- >$ fra le unità visibili e le unità nascoste

Aggiornamento dei pesi:

1. I pesi vengono cambiati secondo la regola: $\Delta w_{ij} = \eta(< v_i^+ h_j^+ > - < v_i^- h_j^- >)$

9.1.5 RBMs vs. Autoencoders

Le RBM e gli autoencoder sono entrambi modelli non supervisionati ma presentano differenze significative:

- RBM è un modello probabilistico generativo, mentre l'autoencoder è un modello discriminativo
- RBM ha connessioni bidirezionali, l'autoencoder ha connessioni unidirezionali
- RBM minimizza la divergenza tra distribuzione dei dati e distribuzione del modello, l'autoencoder minimizza l'errore di ricostruzione

9.1.6 Estrarre Features Visive con RBM

Utilizzando RBM, è possibile estrarre features visive da immagini naturali:

- Apprendimento non supervisionato da un insieme di patches di immagini naturali
- I campi recettivi dei neuroni nascosti sviluppano selettività per caratteristiche specifiche delle immagini (bordi, orientamenti, etc.)
- Queste caratteristiche presentano analogie con i campi recettivi dei neuroni nella corteccia visiva primaria (V1)

9.2 Modelli Generativi Gerarchici

9.2.1 Deep Belief Networks (DBN)

Le Deep Belief Networks sono reti generative profonde che combinano varie RBMs per apprendere modelli interni più complessi:

- Lo strato nascosto di una RBM è usato come input (strato visibile) per la RBM successiva
- In questo modo possiamo apprendere molteplici livelli di rappresentazione (ipotesi su ipotesi)
- Rappresentazioni astratte vengono scoperte in modo non supervisionato, riutilizzando features più semplici
- Ogni strato di neuroni effettua una proiezione non lineare dell'input

Hinton & Salakhutdinov (2006, Science) hanno dimostrato che questo approccio consente di apprendere rappresentazioni gerarchiche efficaci.

9.2.2 Apprendere Rappresentazioni tramite Deep Learning Non Supervisionato

I modelli gerarchici non supervisionati possono apprendere rappresentazioni sempre più astratte dei dati, come dimostrato in vari esperimenti:

- Riconoscimento di cifre manoscritte (MNIST)
- Riconoscimento di volti
- Modellazione di oggetti e scene

9.2.3 Deep Autoencoders

Gli autoencoder profondi possono avere molti strati nascosti:

- Strutture con milioni di neuroni artificiali e miliardi di connessioni sinaptiche
- Possono richiedere giorni di apprendimento intensivo su supercomputer

Anche senza supervisione, la rete è in grado di apprendere features astratte, per esempio il concetto di "volto".

9.3 Generative Adversarial Networks (GAN)

9.3.1 Principio di Funzionamento

Le Generative Adversarial Networks (Goodfellow et al., 2014) sono composte da due reti neurali che competono tra loro in un gioco a somma zero:

- **Generatore**: crea esempi che cercano di sembrare dati reali
- **Discriminatore**: cerca di distinguere tra esempi reali e quelli creati dal generatore

L'obiettivo del modello generativo è di ingannare un classificatore!

9.3.2 Varianti e Miglioramenti

InfoGAN:

- Permette di controllare singole proprietà latenti
- Genera stimoli sensoriali manipolando specifiche caratteristiche

CycleGAN:

- Permette trasformazioni tra domini diversi
- Esempi: cavallo → zebra, foto → stile pittorico

StyleGAN:

- Genera immagini fotorealistiche controllando lo stile
- Utilizzato per creare volti di persone che non esistono (thispersondoesnotexist.com)

9.4 Diffusion Models

9.4.1 Principio di Funzionamento

I Diffusion Models (Song et al., 2021) funzionano in modo diverso dalle GAN:

- Un'immagine di training viene gradualmente trasformata in rumore (tipicamente rumore Gaussiano) tramite un processo di diffusione stocastica
- Il modello generativo (tipicamente una rete convoluzionale o un transformer) apprende il processo stocastico inverso, che a partire dal rumore genera l'immagine
- Questi modelli sono alla base dei generatori di immagine moderni (es: DALL-E, Stable Diffusion)

9.4.2 Modelli Text-to-Image

I modelli Text-to-Image mappano rappresentazioni linguistiche in rappresentazioni visive:

- Un encoder di testo (solitamente un Transformer) converte la descrizione testuale in un embedding
- Questo embedding condiziona il processo di diffusione inversa
- Il risultato è un'immagine che corrisponde alla descrizione testuale

Esempi di questi modelli sono DALL-E, Stable Diffusion e Midjourney.

9.4.3 Progressi e Limitazioni

Con l'aumentare del numero di parametri, la qualità delle generazioni migliora notevolmente:

- DALL-E 3 può generare immagini molto più dettagliate e coerenti rispetto alle versioni precedenti
- I modelli più recenti sono in grado di gestire richieste complesse e concetti astratti

Tuttavia, ci sono ancora limitazioni:

- Difficoltà con alcuni concetti (es: contare correttamente)
- Coerenza imperfetta in scene complesse
- Bias appresi dai dati di training

10. APPRENDIMENTO CON RINFORZO

10.1 Perché Osservare Non Basta: Bisogna Agire

10.1.1 La Prospettiva della "Cognizione Incarnata" (Embodied Cognition)

Finora ci siamo focalizzati su due modalità per percepire ed interpretare l'informazione sensoriale:

- Apprendere compiti di categorizzazione/regressione (apprendimento supervisionato)
- Creare modelli interni scoprendo regolarità statistiche (apprendimento non supervisionato)

Tuttavia, gli agenti cognitivi sono solitamente embodied, ovvero sono dotati di un corpo con effettori (mani, braccia, gambe…) che possono essere usati per manipolare attivamente l'ambiente!

10.1.2 Causalità e Manipolazione Attiva

"Correlation does not imply causation!" - La correlazione non implica causalità.

Per passare dalla scoperta di semplici correlazioni alla scoperta di relazioni causali, dobbiamo manipolare attivamente l'ambiente:

- Esperimenti randomizzati
- Ragionamento controfattuale

10.2 Intuizioni dal Condizionamento Animale

10.2.1 Ispirazione Psicologica

Lo sviluppo del reinforcement learning è stato ispirato dalle teorie psicologiche sull'apprendimento animale:

Condizionamento classico (Ivan Pavlov, 1849-1936):

- La contingenza temporale degli stimoli gioca un ruolo fondamentale nell'apprendimento di associazioni (correlazioni) stimolo-risposta
- Uno stimolo biologicamente saliente (cibo, piacere…) viene accoppiato con uno stimolo a valenza neutra (campanello, luce…)
- Dopo l'apprendimento, lo stimolo neutrale eliciterà la risposta che era normalmente associata allo stimolo saliente

Condizionamento operante (B.F. Skinner, 1904-1990):

- Richiede all'animale di scegliere la risposta più appropriata per ricevere un rinforzo positivo (ed evitare una punizione)
- L'animale impara a massimizzare le ricompense e minimizzare le punizioni

10.3 Apprendere tramite Prove ed Errori

10.3.1 Il Problema Fondamentale

Il problema fondamentale nell'apprendimento con rinforzo è:

> Che azioni dovrei intraprendere per massimizzare i miei guadagni futuri (e minimizzare le punizioni)?

10.3.2 Complicazioni e Possibili Soluzioni

Bilanciamento tra guadagno immediato e a lungo termine:

- Alcune azioni massimizzano il guadagno immediato ma sono controproducenti nel lungo termine
- Altre azioni sembrano inutili, ma si riveleranno importanti in futuro
- **Soluzione**: imparare a predire conseguenze a lungo termine delle proprie azioni

Ambiente stocastico:

- Non possiamo essere sicuri che una particolare azione sia sempre appropriata o dia sempre lo stesso risultato
- **Soluzione**: effettuare scelte probabilistiche

Dilemma esplorazione vs. sfruttamento (Exploration vs. Exploitation):

- Dovrei esplorare nuovi stati (potenzialmente più appaganti, ma potenzialmente anche peggiori) oppure rimanere negli stati che conosco ed "accontentarmi"?
- **Soluzione**: usare procedure simili all'annealing per incoraggiare esplorazione iniziale e poi passare gradualmente alle azioni più consolidate

10.4 Un Framework Matematico per il Reinforcement Learning

10.4.1 Setting Generale

- Set di stati ambientali $S = \{s_1, s_2 \dots s_n\}$
- Set di azioni che si possono effettuare $A = \{a_1, a_2 \dots a_n\}$
- Set di osservazioni sull'ambiente $O = \{o_1, o_2 \dots o_n\}$
- Regole di transizione fra gli stati
- Regole che specificano il guadagno immediato associato con una certa transizione

Ad ogni tempo t, l'agente riceve una nuova osservazione o*t che consiste nello stato s_t e nella ricompensa r_t. Poi sceglie di compiere l'azione a_t dal set di azioni disponibili, che causa una transizione nello stato successivo s*{t+1}, che è associato con la ricompensa r_{t+1}. Il procedimento poi si ripete.

Lo scopo dell'agente è di massimizzare le ricompense accumulate.

10.4.2 Massimizzare la Ricompensa Accumulata

La somma delle ricompense accumulate è chiamata "ritorno":

$$G_t = \sum_{k=0}^{\infty} r_{t+k+1}$$

Solitamente introduciamo un "fattore di sconto" per dare priorità alle ricompense più immediate (scontando quindi quelle distanti nel tempo):

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

dove $\gamma \in [0, 1]$ specifica quanta importanza viene data alle ricompense future.

La presenza di ricompense future rende il compito complesso: è difficile capire quale azione abbia portato alla ricompensa! Questo è noto come il problema di "credit assignment".

10.5 Funzione di Utilità ed Apprendimento Temporal-Difference

10.5.1 Funzione di Utilità

Definiamo funzione di utilità di un certo stato il ritorno atteso associato a tale stato:

$$U(s) = E[G_t | s_t = s]$$

ovvero, la media del ritorno su più episodi quando ci si trova nello stato s.

10.5.2 Equazione di Bellman

L'equazione di Bellman consente di scomporre l'utilità considerando la ricompensa attuale:

$$U(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s, a)[r(s, a, s') + \gamma U(s')]$$

dove:

- $\pi(a|s)$ è la politica, ovvero la probabilità di scegliere l'azione a nello stato s
- $P(s'|s, a)$ è la probabilità di transizione allo stato s' partendo dallo stato s e compiendo l'azione a
- $r(s, a, s')$ è la ricompensa immediata per la transizione da s a s' compiendo l'azione a

10.5.3 Temporal-Difference Learning

Possiamo apprendere un'approssimazione della funzione di utilità sfruttando l'errore di predizione della ricompensa tra due stati consecutivi!

Dopo essere passati al nuovo stato, l'utilità dello stato precedente viene aggiornata in modo da allinearla al valore di ricompensa appena ricevuto:

$$U(s_t) \leftarrow U(s_t) + \alpha[r_{t+1} + \gamma U(s_{t+1}) - U(s_t)]$$

dove:

- α è il learning rate
- $[r_{t+1} + \gamma U(s_{t+1}) - U(s_t)]$ è l'errore di predizione

NB: l'utilità di uno stato dipende anche dall'azione che scegliamo di intraprendere a partire da tale stato!

10.6 Deep Reinforcement Learning

10.6.1 Combinare Deep Learning e Reinforcement Learning

Il Deep Reinforcement Learning combina i principi del reinforcement learning con le capacità di rappresentazione del deep learning:

- Reti neurali profonde vengono utilizzate per approssimare la funzione valore o la politica
- Questo permette di affrontare problemi con spazi di stati e azioni molto grandi o continui

10.6.2 La Rivoluzione del Deep Reinforcement Learning

Il Deep RL ha ottenuto risultati impressionanti in vari domini:

Imparare strategie in giochi:

- DQN (Deep Q-Network) ha imparato a giocare a giochi Atari partendo direttamente dai pixel
- AlphaGo e AlphaZero hanno superato i campioni umani in giochi da tavolo complessi come Go e Scacchi

Controllo motorio:

- Robot che imparano a manipolare oggetti
- Simulazioni di personaggi che imparano a camminare, correre, saltare

Dalla simulazione alla realtà:

- Transfer learning da ambienti simulati a robot reali
- Drone racing e navigazione autonoma

Robot umanoidi "general purpose":

- Robot come quelli di Boston Dynamics o Figure.ai che imparano compiti generici

10.6.3 Multi-agent RL (MARL)

Nella maggior parte dei casi, l'ambiente include altri agenti!

Multi-agent Reinforcement Learning si occupa di scenari in cui più agenti apprendono simultaneamente:

- Cooperazione: gli agenti imparano a lavorare insieme
- Competizione: gli agenti imparano strategie l'uno contro l'altro
- Comunicazione emergente: gli agenti sviluppano spontaneamente protocolli di comunicazione

10.7 L'Apprendimento con Rinforzo nel Cervello

10.7.1 Il Sistema Dopaminergico

I neuroni dopaminergici sembrano codificare l'aspettativa della ricompensa!

Questi neuroni proiettano verso strutture coinvolte nella motivazione e nel comportamento "goal-directed" (es: striatum, nucleus accumbens, corteccia frontale) ma anche verso strutture coinvolte nel controllo emotivo e memoria (es: ippocampo e amigdala).

Queste strutture risultano compromesse in varie patologie:

- Dipendenza da droga
- Gioco d'azzardo
- Comportamento compulsivo
- Parkinson…

10.7.2 Comportamento dei Neuroni Dopaminergici

I neuroni dopaminergici nella scimmia mostrano un comportamento compatibile con la teoria del TD-learning:

- Rispondono con attivazione massima quando viene somministrato uno stimolo appetitoso (es: succo)
- Creando un'associazione con stimoli predittivi, i neuroni dopaminergici si attivano al momento dello stimolo condizionante (CS)
- Se non vengono forniti stimoli predittivi, l'attivazione avviene al momento della Ricompensa (R)
- Se viene somministrato lo stimolo condizionante ma non viene data la ricompensa, l'attività si riduce sotto il livello basale nel momento in cui sarebbe dovuta avvenire la ricompensa

Questo pattern di attivazione è coerente con la codifica dell'errore di predizione della ricompensa nel TD learning!

11. MODELLI COMPUTAZIONALI NELLE SCIENZE COGNITIVE

11.1 L'Obiettivo delle Scienze Cognitive

L'obiettivo delle scienze cognitive è capire come funziona la mente. Questo implica descrivere, spiegare e predire il comportamento umano.

Per questo scopo, la costruzione di teorie verbali non è sufficiente, abbiamo bisogno di modelli quantitativi.

11.2 Tipi di Modelli Cognitivi

I modelli quantitativi possono assumere due forme:

- **Modelli descrittivi**: riassumono i dati in forma matematica, tipicamente utilizzando parametri stimati dai dati
- **Modelli computazionali**: fanno assunzioni sui processi sottostanti e/o sulle rappresentazioni mentali. I parametri e le caratteristiche del modello hanno interpretazioni psicologiche

Piuttosto che offrire una descrizione matematica dei dati, un vero modello cognitivo spiega e predice la cognizione e il comportamento, e i suoi elementi possono essere interpretati in termini psicologici.

11.3 Tassonomia dei Modelli Cognitivi

11.3.1 Classificazione per Formalismo

- **Modelli pre-quantitativi**:
  - Verbali
  - Diagrammatici (box-and-arrows)
- **Modelli quantitativi (matematici)**:
  - Descrittivi
  - Computazionali

11.3.2 Classificazione per Approccio

- **Modelli computazionali simbolici**:
  - Rule-based
  - Probabilistici
- **Modelli computazionali connessionisti (reti neurali)**:
  - Localistici
  - PDP (Parallel Distributed Processing)

11.4 I "Tre Livelli di Analisi" di Marr

Marr (1982) ha proposto tre livelli di analisi per comprendere i sistemi cognitivi:

11.4.1 Livello Computazionale

Quale obiettivo ha la computazione, perché è appropriata, con quale strategia può essere portata a termine? Come risolverebbe il problema un agente ideale o razionale?

11.4.2 Livello Algoritmico

Qual è l'algoritmo appropriato per raggiungere l'obiettivo? Qual è la rappresentazione per l'input e l'output, e l'algoritmo che li mette in relazione?

11.4.3 Livello dell'Implementazione

Come implementare fisicamente rappresentazioni ed algoritmo?

Nelle neuroscienze, i progressi tecnologici e la possibilità di "manipolare" il cervello hanno condotto ad un bias epistemologico verso il livello dell'implementazione (Krakauer et al. 2017, Neuron).

11.5 Modelli e Simulazioni

11.5.1 Scopi di un Modello Computazionale

- Rimpiazzare e/o migliorare modelli verbali
- Sistematizzare i dati sperimentali
- Testare teorie alternative
- Generare predizioni

11.5.2 Dati da Simulare

- Accuratezza e/o tempi di reazione in compiti sperimentali
- Come una funzione cognitiva venga danneggiata in seguito a lesione cerebrale (dati neuropsicologici)
- La traiettoria di acquisizione di una data abilità cognitiva (sviluppo cognitivo tipico e atipico)

11.6 La Modellizzazione Computazionale in Tre Fasi

11.6.1 Da una Teoria ad un Modello Computazionale

- Un programma informatico (software) non "gira" se non è pienamente specificato. Quindi una teoria non può essere implementata in una simulazione al computer se non è pienamente specificata.
- Formulare una teoria sui processi cognitivi attraverso un modello computazionale rivela immediatamente gli aspetti che rendono la teoria incompleta o sotto-specificata.

- Molte assunzioni teoriche potrebbero essere implicite. Renderle esplicite e valutare il loro ruolo è importante per capire i successi e i fallimenti di un modello.

11.6.2 Valutare un Modello Computazionale

Quando il programma "gira" possiamo valutare in modo rigoroso, attraverso la simulazione, l'adeguatezza della teoria: troviamo nel comportamento del modello gli stessi effetti che osserviamo nel comportamento umano?

11.6.3 Discrepanza tra Simulazione e Dati Empirici

La discrepanza tra dati reali e dati simulati ci rivela in che modo la teoria di partenza è sbagliata. A volte una parziale riformulazione di aspetti specifici della teoria possono eliminare le discrepanze – in questo caso la simulazione ha portato ad una migliore teoria. Se le discrepanze sono fondamentali e non possono essere eliminate, la simulazione ha portato a rigettare la teoria.

11.7 Valutare un Modello

11.7.1 Adeguatezza Descrittiva

Il grado di accuratezza con cui un modello predice un set di dati empirici sia a livello qualitativo che quantitativo (altri termini: accuratezza predittiva, fit del modello).

Due modi di valutare l'adeguatezza descrittiva:

- **Qualitativo**: il modello cattura i fenomeni di interesse?
  - Verificare, usando analisi statistiche, se le risposte del modello sono modulate dagli stessi fattori che influenzano la prestazione umana
  - Il modello mostra gli stessi effetti che osserviamo negli studi umani?
  - Il modello mostra effetti paradossali che non osserviamo negli studi umani?
- **Quantitativo**: quanto è buono il fit (adattamento) ai dati empirici?
  - Misurare l'adattamento del modello (goodness of fit), ad esempio la proporzione di varianza spiegata (R-quadrato) quando i dati del modello sono utilizzati come predittori dei dati umani in un'analisi di regressione lineare

Importante: MODELLO ADEGUATO ≠ MODELLO GIUSTO!

Obiettivi nel trasformare una teoria in modello computazionale:

- **Completezza**: tutti i processi sono stati adeguatamente specificati
- **Sufficienza**: il modello offre una spiegazione di tutti i fenomeni empirici rilevanti

MA questo non garantisce che la teoria sia corretta! È possibile infatti che una seconda teoria, implementata in un modello computazionale, produca lo stesso risultato.

11.7.2 Altri Criteri di Valutazione

Generalità: Un modello non dovrebbe limitarsi a spiegare un effetto specifico, misurato in un solo compito e/o con una singola variabile dipendente. Dovrebbe essere in grado di rendere conto di più fenomeni e simularli attraverso variazioni del set di stimoli e del tipo di compito o di risposta.

Semplicità: È l'opposto della complessità di un modello, che viene di solito indicata dal numero di parametri "liberi" (modificabili) o dalla complessità dell'architettura. La semplicità può essere valutata più facilmente in termini relativi che assoluti. Un modello può essere più semplice rispetto a modelli alternativi, oppure essere relativamente semplice rispetto al grado di adeguatezza descrittiva e di generalità.

11.7.3 Come Aggiudicare tra Teorie in Competizione?

L'approccio computazionale facilita l'aggiudicazione. Se le teorie in competizione sono davvero diverse, lo saranno anche i rispettivi modelli computazionali.

Fase 1: confrontare l'adeguatezza descrittiva dei diversi modelli

- Ci sono effetti critici spiegati da un modello ma non dall'altro?
- Un modello spiega una proporzione maggiore di varianza nei dati umani?

Fase 2: se due modelli sono molto simili dal punto di vista dell'adeguatezza descrittiva, come possiamo scegliere tra i due?

- Criterio della semplicità: scegliere il modello più semplice
- Falsificazione: individuare una divergenza tra predizioni dei modelli e testarle attraverso uno studio sperimentale su soggetti umani

11.8 Modelli Connessionisti

11.8.1 Modelli Connessionisti Localistici

- Sono modelli della "prestazione", non dell'apprendimento (di solito non presente)
- Ogni nodo della rete ha un ruolo predefinito e codifica "localmente" informazioni specifiche (rappresentazioni localistiche)
- Tutti i livelli di rappresentazione vengono decisi a priori
- Enfasi sulla simulazione di dati comportamentali

Esempi:

- Pandemonium (Selfridge, 1959): attivazione in parallelo, solo connessioni feed-forward eccitatorie
- Logogen (Morton, 1969): integrazione di più sorgenti di attivazione; soglia di attivazione riflette frequenza lessicale

- Interactive Activation Model (McClelland e Rumelhart, 1981): interattività tra livelli (con feedback top-down) e connessioni inibitorie tra unità (inibizione laterale)

11.8.2 Modelli Connessionisti PDP (Parallel Distributed Processing)

- Enfasi sull'apprendimento di una capacità (o di un compito)
- Le rappresentazioni sono spesso distribuite su molti nodi della rete
- Alcuni livelli di rappresentazione possono emergere nella rete durante l'apprendimento senza essere definiti a priori (attività strati nascosti)
- Funzioni cognitive come proprietà emergenti della rete neurale

11.9 Esempi di Modelli Computazionali

11.9.1 Modello della Lettura di Seidenberg & McClelland (1989)

Questo modello connessionista PDP è uno dei più influenti nel campo della lettura. Mostra come una rete neurale può apprendere a mappare tra ortografia, fonologia e semantica.

11.9.2 Limitazioni dei Modelli PDP "Classici"

I modelli connessionisti (PDP) "classici" (da 1986 fino a 2010 circa) avevano una o più delle seguenti caratteristiche non desiderabili:

- Architettura feed-forward (no connessioni feedback)
- Architettura "superficiale" (non più di uno strato nascosto)
- Apprendimento discriminativo (supervisionato), che assume la presenza di un insegnante esterno anche quando psicologicamente implausibile
- Input non realistico e/o su piccola scala ("toy models")

Queste limitazioni sono state superate dai modelli basati su deep learning generativo.

11.9.3 Un Framework Probabilistico per i Modelli Connessionisti

L'apprendimento generativo (non supervisionato) è un ingrediente fondamentale per modelli plausibili e su larga scala della percezione e della cognizione umana:

- Apprendimento per osservazione: l'obiettivo è di costruire un modello interno (cause latenti) dell'informazione sensoriale
- Non serve supervisione o rinforzo: diversamente dall'apprendimento discriminativo, non c'è un compito e non ci sono informazioni su cosa è contenuto nell'input (dati non etichettati)

- Approccio probabilistico: date le variabili osservate (evidenza) vengono formulate delle ipotesi (variabili latenti); vanno trovati i parametri del modello che meglio descrivono le osservazioni

Elaborazione gerarchica e ricorrente: l'apprendimento generativo nelle reti neurali richiede elaborazione ricorrente (connessioni a feedback) ed è particolarmente potente quando ci sono molti strati di neuroni organizzati gerarchicamente (deep networks).

11.9.4 Deep Learning per la Percezione delle Lettere

Un esempio di modello cognitivo con deep learning è quello di Testolin, Stoianov & Zorzi (2017, Nature Human Behaviour) per il riconoscimento di lettere:

- Il modello dovrebbe apprendere a riconoscere lettere presentate come immagini reali in una varietà di font, stili, dimensioni e allineamenti spaziali
- Features visive vengono apprese da patch di immagini naturali
- Queste features possono essere "riciclate" per apprendere simboli visivi
- Il modello spiega vari dati psicofisici umani sulla percezione di lettere

12. PROSPETTIVE E SFIDE PER L'IA

12.1 Verso l'Intelligenza Artificiale Generale (AGI)

Ottenere un'IA generale a livello umano è attualmente l'obiettivo esplicito di alcune delle più importanti aziende di AI (OpenAI, DeepMind, Anthropic).

Il concetto di Artificial General Intelligence (AGI) è utilizzato per descrivere un sistema IA che ha come minimo capacità simili a quelle umane in un'ampia gamma di compiti.

Alcuni ricercatori sostengono che ci sono "scintille" di AGI nella più recente generazione di Large Language Models (LLM), es. GPT-4. La nozione di AGI è strettamente collegata a quella di abilità "emergenti" che non erano esplicitamente anticipate durante lo sviluppo del modello.

La definizione stessa di AGI e la sua valutazione rimane problematica:

- Turing test: si basa sulla capacità specifica dell'IA di ingannare un umano piuttosto che su una valutazione ad ampio raggio della sua intelligenza
- Prestazione a livello umano in compiti cognitivi: questa definizione rimane ambigua dal punto di vista della scelta dei compiti e dei benchmark umani

12.2 I Rischi dell'IA

L'adozione diffusa dell'IA comporta vari rischi:

- Abusi nell'utilizzo (es. biometria in tempo reale)

- Responsabilità (anche civile: chi paga per danni da IA?)
- Minacce ai diritti fondamentali (es. discriminazione, protezione dei dati e della vita privata)
- Effetti sul mondo del lavoro (14% posti di lavoro automatizzabili + 32% affrontano cambiamenti sostanziali – fonte Parlamento UE)
- Distorsioni della concorrenza
- Cyber sicurezza
- Problemi di trasparenza (interagisco con umano o IA?)

12.3 Le Sfide "Etiche" per l'IA

12.3.1 Spiegabilità dell'IA (Explainable AI)

La spiegabilità dell'IA si riferisce alla capacità di spiegare (trasparenza) e giustificare le scelte (responsabilità) di un sistema di IA.

È particolarmente critica in ambiti come:

- Medicina (diagnosi e terapie)
- Finanza (decisioni di credito)
- Giustizia (valutazione del rischio)
- Risorse umane (selezione del personale)

12.3.2 Assenza di Pregiudizio (AI Fairness)

L'AI Fairness si concentra sul controllare e prevenire bias nelle risposte/decisioni dell'IA.

Il sistema COMPAS per la predizione della recidiva negli USA ha mostrato come i bias possono essere nascosti nei dati utilizzati per l'addestramento, portando a discriminazioni razziali nelle predizioni.

12.3.3 Allineamento al Giudizio Morale Umano

Le scelte e decisioni dei sistemi di IA devono essere moralmente accettabili secondo i valori umani.

Un esempio di dilemma morale in ambito IA per guida autonoma: in uno scenario di guasto ai freni, l'inazione causa la morte di tre persone anziane che stanno attraversando (con il rosso), mentre sterzare causa la morte dei tre passeggeri (una famiglia).

12.4 Human-centered AI

Concetto chiave: Human-in-the-Loop

- L'umano è centrale, dal design dell'IA al suo controllo e utilizzo (no IA autonoma)

- Enfasi sull'aumentare le capacità umane piuttosto che sul sostituirle
- Enfasi sull'etica dell'IA (imparzialità, responsabilità, trasparenza)
- Lo sviluppo richiede team multidisciplinari

Gli stessi principi sono alla base del "Regolamento UE sull'Intelligenza Artificiale" (AI Act).

12.5 Responsabilità nell'IA

La responsabilità nell'ambito dell'IA può assumere diverse forme:

- Responsabilità per le proprie azioni e i loro effetti (accountability)
- Responsabilità di ruolo
- Responsabilità morale
- Responsabilità passiva: ricostruzione delle responsabilità dopo un evento negativo
- Responsabilità attiva: cercare un impatto positivo ed agire in modo proattivo per prevenire effetti negativi
    - Ethical-by-design (design della tecnologia incorpora valori etici)

12.6 Trasparenza e Diritto alla Spiegazione

Due fattori critici per mantenere fiducia nei sistemi IA, assicurarne l'utilizzo responsabile, e prevenire conseguenze legali:

- **Trasparenza**: tutti gli attori (organizzazione, utente dell'IA, persone su cui l'IA ha impatto) dovrebbero avere informazioni sul funzionamento dell'IA, sui dati raccolti, su quali fattori guidano le decisioni
- **Spiegabilità**: l'IA deve fornire spiegazioni per decisioni individuali che siano comprensibili agli utenti

12.7 Il Regolamento UE sull'IA

12.7.1 Obiettivi del Regolamento

- Sicurezza dei sistemi AI, rispetto dei diritti fondamentali e dei valori dell'Unione
- Governance e certezza del diritto per facilitare investimenti e innovazione
- Sistemi sicuri, trasparenti, tracciabili, non discriminatori e rispettosi dell'ambiente

12.7.2 Definizione di Intelligenza Artificiale

Un sistema progettato per operare con vari livelli di autonomia che può, per obiettivi espliciti o impliciti, generare risultati come previsioni, raccomandazioni o decisioni che influenzano ambienti fisici o virtuali.

12.7.3 Classificazione del Rischio

IA a rischio inaccettabile:

- Sistemi vietati perché costituiscono una minaccia per le persone
- Esempi: manipolazione comportamentale cognitiva di persone vulnerabili, classificazione sociale, sistemi di identificazione biometrica in tempo reale

IA ad alto rischio:

- Sistemi capaci di incidere in modo sensibile sulla salute e sui diritti fondamentali
- Esempi: infrastrutture critiche, identificazione biometrica, istruzione, occupazione, servizi essenziali

IA a rischio limitato:

- Sistemi con obblighi di trasparenza
- Esempi: chatbot, deepfake, contenuti sintetici

12.8 Sostenibilità dell'IA

Una sfida ulteriore è la sostenibilità energetica dell'IA:

- La potenza di calcolo necessaria per lo sviluppo dell'IA sta raddoppiando ogni 100 giorni circa
- L'energia necessaria per l'IA è in forte crescita, con un tasso di incremento annuo compreso tra il 26% e il 36%
- L'IA nel 2028 potrebbe consumare più energia di quanta ne abbia utilizzata l'intera Islanda nel 2021

12.9 Conclusioni

- La "rivoluzione" del deep learning ha prodotto tecnologie cognitive con prestazioni pari a quelle umane in problemi specifici e circoscritti. L'IA generale (AGI) è attualmente un obiettivo esplicito ma non ancora raggiunto.
- L'approccio "human-in-the-loop" (human-centered AI) è attualmente la migliore opzione per un approccio "ethical-by-design" e per essere in linea con la regolamentazione europea sull'IA (AI Act).
- L'interpretabilità e la trasparenza dell'IA è attualmente uno dei maggiori fattori limitanti per l'utilizzo su larga scala del deep learning in sistemi di decisione, soprattutto negli ambiti definiti a rischio elevato dall'AI Act.
- La sinergia tra IA e intelligenza umana si può realizzare spostando l'enfasi dall'automatizzazione e dalla sostituzione del "fattore umano" - guardando all'IA non (solo) come assistente digitale ma come leva per aumentare le nostre capacità cognitive.