

Dietro le quinte dell'IA generativa: come nasce una vettorizzazione

DI [MASSIMO VAILATI](#)

22/10/2026

INTELLIGENZA ARTIFICIALE

SCIENTIFICO S.A., TT INFORMATICA



L'intelligenza artificiale generativa utilizza una tecnica chiamata *embedding* o vettorizzazione per trasformare parole e frasi in vettori numerici, permettendo ai modelli linguistici di riconoscere le relazioni tra i significati dei testi. Attraverso semplici esempi e attività pratiche in Python, gli studenti scopriranno come calcolare la distanza tra parole e confrontare frasi utilizzando la similarità del coseno. Questi principi permettono di sviluppare motori di ricerca semantici, capaci di individuare informazioni pertinenti anche quando le parole utilizzate sono diverse.

L'approfondimento sulla tecnologia RAG mostrerà, infine, come l'IA possa consultare documenti e fonti esterne per generare risposte più precise e aggiornate.

I computer sono calcolatori eccezionali, ma soffrono di un grande limite: non capiscono le parole. Per un processore, la stringa "Cane" è solo una sequenza arbitraria di caratteri ASCII o UTF-8 (01000011 01100001...).

Se chiedessimo a un algoritmo base se "Cane" è più simile a "Gatto" o a "Aereo", basandosi solo sulle lettere direbbe che nessuna di queste si assomiglia. Per la macchina sono tutte stringhe diverse.

Come fa allora ChatGPT o in generale un LLM (Large Language Model) a sapere che un cane e un gatto sono entrambi animali domestici a quattro zampe? La risposta sta nei vettori e nel processo chiamato **embedding** (vettorizzazione).

L'embedding è la mappa concettuale dell'IA. Immagina di prendere ogni parola del vocabolario e posizionarla all'interno di uno spazio geometrico.

Se assegniamo a ogni parola delle "coordinate" basate sulle sue caratteristiche:

Dimensione 1 (Animale): da 0.0 (non animale) a 1.0 (estremamente animale)

Dimensione 2 (Taglia): da 0.0 (minuscolo) a 1.0 (gigante)

Dimensione 3 (Capacità di volare): da 0.0 (no) a 1.0 (sì)

Otteniamo dei vettori in tre dimensioni:

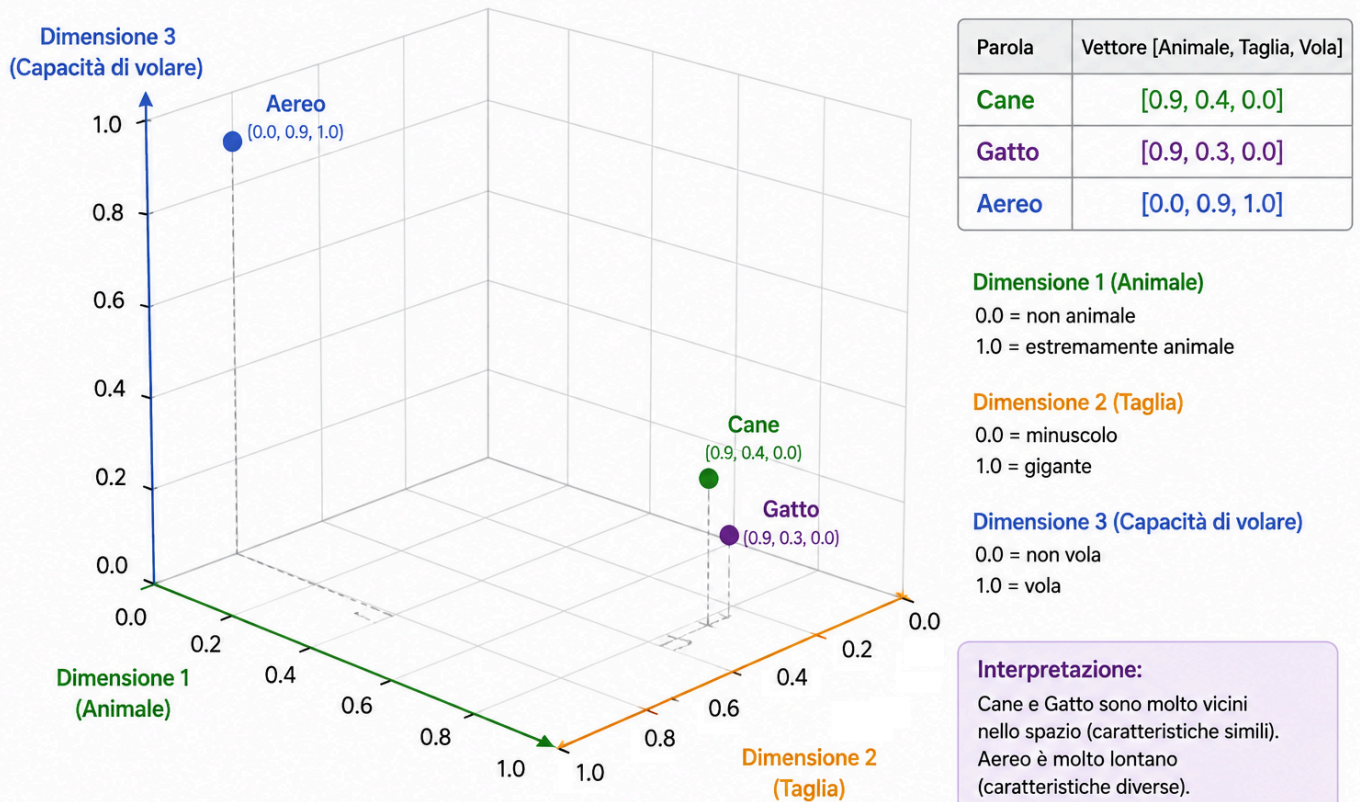
{Cane} = [0.9, 0.4, 0.0]

{Gatto} = [0.9, 0.3, 0.0]

{Aereo} = [0.0, 0.9, 1.0]

I punti relativi a Cane e Gatto saranno vicinissimi tra loro nello spazio, mentre Aereo sarà posizionato molto lontano. L'IA non "legge" il testo, ma misura la distanza matematica tra questi punti. Nei modelli reali (come quelli di OpenAI o Google) le dimensioni non sono 3, ma centinaia o migliaia.

Rappresentazione 3D delle parole nello spazio delle caratteristiche



Attività pratica n.1

Calcola la distanza delle parole

Questa attività mostra agli studenti come la matematica calcola la similarità semantica senza bisogno di usare librerie AI complesse.

Codice Python da proporre agli studenti

```
import math

# 1. Definiamo uno spazio a 3 dimensioni: [Animale, Taglia, Volante]
parole_vettori = {
    "Cane": [0.9, 0.4, 0.0],
    "Gatto": [0.9, 0.3, 0.0],
    "Aquila": [0.8, 0.2, 1.0],
    "Aereo": [0.0, 0.9, 1.0]
}
```

```
def distanza_euclidea(v1, v2):  
    """Calcola la distanza geometrica tra due vettori."""  
    somma_quadrati = sum((a - b) ** 2 for a, b in zip(v1, v2))  
    return math.sqrt(somma_quadrati)  
  
# 2. Confrontiamo la distanza tra 'Cane' e le altre parole  
parola_target = "Cane"  
vettore_target = parole_vettori[parola_target]  
  
print(f"--- Distanze dalla parola '{parola_target}' ---")  
for parola, vettore in parole_vettori.items():  
    if parola != parola_target:  
        dist = distanza_euclidea(vettore_target, vettore)  
        print(f"Distanza da {parola:7s}: {dist:.3f}")
```

Output del programma

```
--- Distanze dalla parola 'Cane' ---  
Distanza da Gatto : 0.100  
Distanza da Aquila : 1.025  
Distanza da Aereo : 1.435
```

Osserviamo come la distanza "Cane-Gatto" risulti minima (circa 0.100), mentre "Cane-Aereo" risulti molto grande (oltre 1.400). A questo punto potremmo chiedere agli studenti di aggiungere una quarta dimensione (es. Domestico) e modificare i valori per vedere come cambiano le distanze.

Nelle **reti neurali** reali, le coordinate non vengono inserite a mano dagli umani, ma vengono "imparate" dall'IA analizzando miliardi di testi (notando quali parole compaiono spesso nello stesso contesto).

Approfondimento: dalla teoria al progetto "Search Engine"



Per completare l'articolo non ci limiteremo a calcolare la distanza tra singoli concetti, ma vedremo come questi vettori permettono di costruire anche i motori di ricerca di un'IA. La **Retrieval-Augmented Generation (RAG)** è una tecnologia che migliora le prestazioni dei modelli linguistici di grandi dimensioni (LLM), consentendo loro di consultare fonti di conoscenza esterne e aggiornate prima di generare una risposta. A differenza degli LLM tradizionali, che si basano esclusivamente sui dati utilizzati durante l'addestramento, la RAG integra informazioni provenienti da database, documenti aziendali o altre basi di conoscenza autorevoli, senza richiedere il riaddestramento del modello.

Il funzionamento della RAG si basa su un sistema di recupero delle informazioni: la richiesta dell'utente viene trasformata in una rappresentazione vettoriale e confrontata con un database vettoriale per individuare i contenuti più pertinenti. Le informazioni recuperate vengono quindi fornite al modello linguistico insieme alla domanda originale, permettendogli di elaborare risposte più precise, aggiornate e contestualizzate.

Nell'esempio base abbiamo usato la **Distanza Euclidea** (il Teorema di Pitagora esteso). Nei modelli reali di Text Embedding, tuttavia, si preferisce la **Similitudine del Coseno** (*Cosine Similarity*).

Se confrontiamo una frase breve ("*Gatto felice*") con un paragrafo lungo che parla di gatti, la distanza euclidea potrebbe risultare enorme a causa della diversa lunghezza del testo (magnitudo del vettore).

La similitudine del coseno misura l'**angolo** formato dai due vettori nello spazio, ignorando la loro lunghezza:

- **Angolo 0° (cos = 1):** I testi sono concettualmente identici.
- **Angolo 90° (cos = 0):** I testi non hanno alcuna relazione semantica.
- **Angolo 180° (cos = -1):** I testi sono uno l'opposto dell'altro.

La formula matematica per due vettori **A** e **B** è:

$$\text{Similarity}(A, B) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Attività pratica n. 2

Costruiamo un Mini-Motore di Ricerca Semantico

Codice Python da proporre agli studenti

```
import math

# Database di frasi (i nostri 'documenti') vettorizzati su 4 dimensioni:
# [Informatica/Tech, Natura/Animali, Cibo/Cucina, Emozioni/Sentimenti]
documenti = {
    "Doc1 - Il gatto dorme sul divano": [0.1, 0.9, 0.1, 0.4],
    "Doc2 - Python è un linguaggio potente": [1.0, 0.0, 0.0, 0.2],
    "Doc3 - La ricetta della pizza margherita": [0.0, 0.0, 1.0, 0.5],
    "Doc4 - Programmare hardware in C++": [0.9, 0.0, 0.0, 0.1],
    "Doc5 - Adoro passeggiare nei boschi": [0.0, 0.7, 0.3, 0.8]
}

def cosine_similarity(v1, v2):
    dot_product = sum(a * b for a, b in zip(v1, v2))
    norm_v1 = math.sqrt(sum(a ** 2 for a in v1))
    norm_v2 = math.sqrt(sum(b ** 2 for b in v2))
    if norm_v1 == 0 or norm_v2 == 0:
        return 0.0
    return dot_product / (norm_v1 * norm_v2)

# L'utente cerca una frase che NON contiene le parole esatte dei documenti
# Query: "Scrivere codice per computer" -> Vettore stimato:
```

```
query_vettore = [0.95, 0.0, 0.0, 0.1] # Alta affinità con Informatica/Tech, bassa con
le altre dimensioni

# Calcolo della similarità per ogni documento
risultati = []
for doc, vettore in documenti.items():
    sim = cosine_similarity(query_vettore, vettore)
    risultati.append((doc, sim))

# Ordiniamo i risultati dal più pertinente al meno pertinente
risultati.sort(key=lambda x: x[1], reverse=True)

print("--- RISULTATI DELLA RICERCA SEMANTICA ---")
for doc, sim in risultati:
    print(f"Affinità: {sim*100:5.1f}% | {doc}")
```

Output del programma

```
--- RISULTATI DELLA RICERCA SEMANTICA ---
Affinità: 100.0% | Doc4 - Programmare hardware in C++
Affinità: 99.6% | Doc2 - Python è un linguaggio potente
Affinità: 14.2% | Doc1 - Il gatto dorme sul divano
Affinità: 7.6% | Doc5 - Adoro passeggiare nei boschi
Affinità: 4.7% | Doc3 - La ricetta della pizza margherita
```

Perché una ricerca per la parola "codice" fallirebbe sul Doc4 se usassimo una ricerca testuale classica, mentre il nostro algoritmo basato su vettori lo posiziona al primo posto?

Ogni frase viene rappresentata come un **embedding**, cioè un vettore numerico che descrive il significato della frase.

Le quattro dimensioni rappresentano:

1. Informatica/Tech
2. Natura/Animali
3. Cibo/Cucina
4. Emozioni/Sentimenti

Il vettore della query [0.95, 0.0, 0.0, 0.1] e il vettore del Doc4 [0.9, 0.0, 0.0, 0.1] indicano una frase fortemente legata all'informatica: sono praticamente nella **stessa direzione** nello

spazio vettoriale. La similarità coseno risulta infatti circa **100%**, perché i due vettori rappresentano concetti quasi identici.

In altre parole, il sistema “capisce” che:

- scrivere codice
- programmare
- C++
- sviluppo software

sono concetti molto vicini, anche se usano parole diverse.