

Large Language Models

Architettura, Addestramento e Impatti Applicativi

[Il tuo nome]

Anno Accademico 2024–2025

Indice

1	Introduzione	1
1.1	Introduzione	1
2	Autoencoders	3
2.1	Introduzione	3
2.2	Autoencoders: definizione e formulazione generale	3
2.2.1	Apprendimento non supervisionato	3
2.2.2	Encoder, decoder e spazio latente	4
2.2.3	Funzione obiettivo e errore di ricostruzione	5
2.3	Il problema dell'identità e la necessità di vincoli	5
2.3.1	Bottleneck e riduzione della dimensionalità	6
2.3.2	Introduzione di vincoli	6
2.3.3	Relazioni con la PCA	8
2.4	Interpretabilità delle feature latenti	12
2.4.1	Rappresentazioni latenti disentangled	13
2.5	Sparse Autoencoders	14
3	Embeddings	17
3.1	Introduzione	18
3.2	L'ipotesi distribuzionale	18
3.3	Ipotesi di Osgood	19

3.4	Embeddings	20
3.4.1	Embeddings count-based	21
3.4.2	Riduzione dimensionale tramite SVD	24
3.4.3	Cosine Similarity	25
3.4.4	Word2Vec: un approccio predittivo	26
3.4.5	Proprietà semantiche degli embeddings	28
3.5	Contextual Embeddings	30
3.6	Modelli di linguaggio neurale	31
3.7	Reti neurali ricorrenti e LSTM	33
3.8	ELMo e il contesto bidirezionale	35
3.9	Attention e Self-Attention	36
3.10	BERT e Masked Language Modeling	37
3.11	Motivazione per il disentanglement degli embeddings di BERT	39
3.12	Pooling	40
3.12.1	Pooling gerarchico per sequenze lunghe	43
4	Disentangling Dense Embeddings with Sparse Autoencoders	45
4.1	Motivazione e contesto	45
4.2	Metodologia e Architettura	46
4.2.1	Definizione del Modello	46
4.2.2	Vincolo di Sparsità <i>k-Sparse</i>	47
4.2.3	Funzione di Costo e Addestramento	47
4.3	Interpretazione Automatizzata delle Feature	48
4.4	Feature Families e Struttura Gerarchica	48
4.5	Feature Families e Struttura Gerarchica	49
4.5.1	Costruzione del Grafo di Co-occorrenza	50
4.5.2	Identificazione delle Feature Families	51

5	Prisma	53
5.1	Introduzione	53
5.2	Architettura	53
5.3	Generazione Embeddings	54
5.3.1	Gestione di documenti lunghi: strategia <i>chunk-and-average</i>	54
5.4	Training SAE	56
5.5	Interpretazione	57
6	Esperimenti e risultati	59
6.1	Introduzione	59
6.2	Pedianet	59
6.2.1	Scelta del modello di embedding	60
6.2.2	Esperimento	60
6.3	Abstracts	61
6.4	PubMed	61

Capitolo 1

Introduzione

1.1 Introduzione

Il 30 Novembre 2022, con l'avvento di ChatGPT è stata segnata una data storica per l'umanità, non solo nel campo della tecnologia ma anche nell'ambito filosofico in quanto per la prima volta l'uomo ha iniziato a parlare con qualcosa di altro da sé che sembra dar prova che il linguaggio umano, veicolo di significato, non sia esclusivo dell'uomo ma possa essere appreso e riprodotto da una macchina.

Capitolo 2

Autoencoders

2.1 Introduzione

In questo capitolo vengono introdotti gli *autoencoders*, una classe di modelli di apprendimento non supervisionato ampiamente utilizzata per l'apprendimento di rappresentazioni latenti dei dati. Dopo averne presentato la formulazione generale e i principi di funzionamento, verranno discussi i principali limiti degli autoencoders classici, in particolare in termini di capacità di apprendere rappresentazioni interpretabili.

Successivamente, il capitolo introduce gli *Sparse Autoencoders*, una estensione degli autoencoders tradizionali che impone vincoli di sparsità sullo spazio latente, favorendo il disentanglement delle feature e l'interpretabilità delle rappresentazioni apprese. Questi modelli costituiscono il fondamento teorico delle metodologie utilizzate nel resto del lavoro di tesi.

2.2 Autoencoders: definizione e formulazione generale

2.2.1 Apprendimento non supervisionato

Gli autoencoders sono modelli di apprendimento non supervisionato, in quanto non richiedono etichette associate ai dati di input durante la fase di addestramento. Si consideri un dataset di addestramento S_T costituito da M

osservazioni non etichettate \mathbf{x}_i , con $i = 1, \dots, M$:

$$S_T = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M\}. \quad (2.1)$$

In generale, ciascuna osservazione appartiene allo spazio \mathbb{R}^n , ovvero $\mathbf{x}_i \in \mathbb{R}^n$. L'obiettivo di un autoencoder è apprendere una rappresentazione dei dati tale da permettere la ricostruzione dell'input nel modo più accurato possibile, minimizzando una misura dell'errore di ricostruzione. L'interesse verso questo tipo di modelli risiede nel fatto che la rappresentazione latente appresa può essere utilizzata in numerose applicazioni, come la riduzione della dimensionalità, l'estrazione di caratteristiche, il denoising e l'anomaly detection. Una definizione formale di autoencoder è la seguente.

Autoencoder *Un autoencoder è un tipo di algoritmo il cui scopo principale è apprendere una rappresentazione dei dati, utilizzabile per diverse applicazioni, imparando a ricostruire in modo sufficientemente accurato un insieme di osservazioni di input [1].*

2.2.2 Encoder, decoder e spazio latente

Un autoencoder è composto da due blocchi principali: un **encoder** e un **decoder**. La struttura generale del modello è illustrata in Figura 2.1.

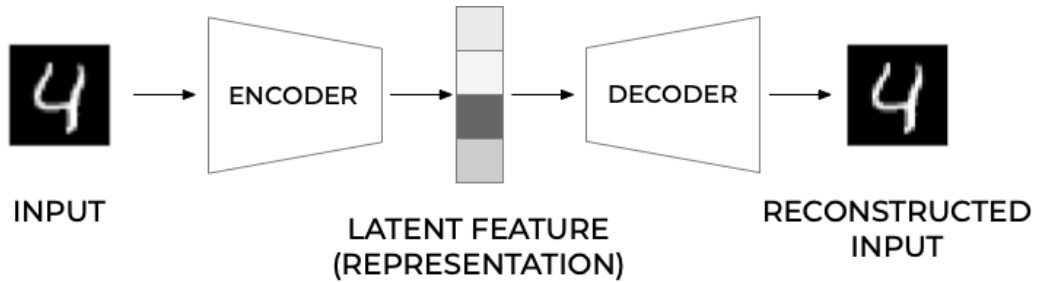


Figura 2.1: Schema di funzionamento di un autoencoder [2]

Nella maggior parte dei casi, l'encoder e il decoder sono implementati come reti neurali. Seguendo l'impostazione descritta in [2], l'encoder può essere rappresentato come una funzione g , dipendente da un insieme di parametri apprendibili, che associa a ciascun dato di input una rappresentazione nello spazio latente:

$$\mathbf{h}_i = g(\mathbf{x}_i). \quad (2.2)$$

2.3. IL PROBLEMA DELL'IDENTITÀ E LA NECESSITÀ DI VINCOLI

Qui $\mathbf{h}_i \in \mathbb{R}^q$ rappresenta il vettore delle *features latenti* ed è l'output del blocco di encoder quando la funzione g viene valutata sull'input \mathbf{x}_i . Ne consegue che l'encoder realizza una mappatura del tipo

$$g : \mathbb{R}^n \rightarrow \mathbb{R}^q. \quad (2.3)$$

Il decoder ha il compito di ricostruire il dato originale a partire dalla rappresentazione latente. L'output della rete, indicato con $\hat{\mathbf{x}}_i$, può essere espresso tramite una seconda funzione generica f :

$$\hat{\mathbf{x}}_i = f(\mathbf{h}_i) = f(g(\mathbf{x}_i)), \quad (2.4)$$

dove $\hat{\mathbf{x}}_i \in \mathbb{R}^n$ rappresenta la ricostruzione dell'input \mathbf{x}_i .

2.2.3 Funzione obiettivo e errore di ricostruzione

L'addestramento di un autoencoder consiste nel determinare le funzioni $g(\cdot)$ e $f(\cdot)$ tali da minimizzare una misura della discrepanza tra i dati di input e le rispettive ricostruzioni. Formalmente, il problema di ottimizzazione può essere espresso come

$$\arg \min_{f,g} \langle \Delta(\mathbf{x}_i, f(g(\mathbf{x}_i))) \rangle, \quad (2.5)$$

dove Δ indica una funzione di perdita che quantifica la differenza tra l'input e l'output dell'autoencoder, mentre $\langle \cdot \rangle$ denota la media su tutte le osservazioni del dataset di addestramento.

2.3 Il problema dell'identità e la necessità di vincoli

In assenza di vincoli sull'architettura o sulla funzione obiettivo, un autoencoder dotato di capacità sufficiente può apprendere una semplice funzione identità, ottenendo una ricostruzione perfetta ma priva di utilità pratica. Per evitare questo comportamento degenerato, è comune introdurre specifiche strategie di regolarizzazione, come la presenza di una strozzatura dimensionale nello spazio latente oppure l'aggiunta di termini di regolarizzazione alla funzione di costo.

Nota. Un autoencoder efficace deve bilanciare due obiettivi contrastanti: da un lato una ricostruzione sufficientemente accurata dell'input, dall'altro l'apprendimento di una rappresentazione latente che catturi le caratteristiche essenziali dei dati, evitando soluzioni banali come l'identità.

2.3.1 Bottleneck e riduzione della dimensionalità

Al fine di evitare che l'autoencoder apprenda una banale funzione identità e di favorire l'apprendimento di rappresentazioni astratte e informative dei dati, una strategia comunemente adottata consiste nell'imporre una riduzione della dimensionalità tra lo spazio di input e lo spazio latente. Tale configurazione architetturale prende il nome di **bottleneck** o **strozzatura**. In un'architettura con bottleneck, la dimensione dello spazio latente q è strettamente inferiore alla dimensione dell'input n ($q < n$). In queste condizioni, l'encoder realizza una mappatura che comprime l'informazione contenuta nei dati di ingresso:

$$g : \mathbb{R}^n \rightarrow \mathbb{R}^q, \quad q < n, \quad (2.6)$$

costringendo il modello a selezionare e preservare esclusivamente le componenti più rilevanti dell'input ai fini della ricostruzione. La presenza della strozzatura impedisce quindi una copia diretta dei dati e spinge l'autoencoder a catturare strutture, correlazioni e regolarità latenti presenti nel dataset. Al termine dell'addestramento, lo spazio latente costituisce una rappresentazione compatta e astratta dei dati, che può essere interpretata come una codifica delle caratteristiche essenziali dell'input e utilizzata per compiti successivi quali riduzione della dimensionalità, visualizzazione o analisi delle feature.

2.3.2 Introduzione di vincoli

Oltre alla strozzatura architetturale, un ulteriore approccio per evitare che l'autoencoder apprenda una semplice funzione identità consiste nell'introduzione di vincoli aggiuntivi nella funzione obiettivo, tipicamente sotto forma di termini di regolarizzazione. Tali vincoli agiscono limitando la capacità espressiva del modello o penalizzando soluzioni considerate indesiderabili, favorendo l'apprendimento di rappresentazioni latenti più strutturate e informative.

2.3. IL PROBLEMA DELL'IDENTITÀ E LA NECESSITÀ DI VINCOLI 17

In questo contesto, la funzione di costo dell'autoencoder non si limita più a misurare esclusivamente l'errore di ricostruzione, ma include uno o più termini addizionali che impongono specifiche proprietà alla rappresentazione latente o ai parametri del modello. In forma generale, il problema di ottimizzazione può essere scritto come

$$\arg \min_{f,g} \langle \Delta(\mathbf{x}_i, f(g(\mathbf{x}_i))) \rangle + \lambda \Omega(g, f), \quad (2.7)$$

dove $\Omega(g, f)$ rappresenta un termine di regolarizzazione e $\lambda > 0$ ne controlla l'importanza relativa rispetto all'errore di ricostruzione. A seconda della scelta del termine di regolarizzazione, è possibile indurre diverse proprietà nel modello. Ad esempio, la penalizzazione della norma dei pesi limita la complessità della rete e migliora la capacità di generalizzazione, mentre vincoli applicati direttamente allo spazio latente possono favorire caratteristiche quali la **sparsità**, la robustezza al rumore o la separazione delle feature. In particolare, l'introduzione di vincoli di sparsità sulle attivazioni latenti costituisce il principio alla base degli *Sparse Autoencoders*, che verranno discussi nel seguito.

Un esempio comune di regolarizzazione consiste nell'introdurre un vincolo direttamente sulle attivazioni dello spazio latente. In questo caso, la funzione obiettivo dell'autoencoder assume la forma

$$\arg \min_{f,g} \langle \Delta(\mathbf{x}_i, f(g(\mathbf{x}_i))) \rangle + \lambda \|\mathbf{h}_i\|_2^2, \quad (2.8)$$

dove $\mathbf{h}_i = g(\mathbf{x}_i)$ denota il vettore delle attivazioni latenti associate all'osservazione \mathbf{x}_i . Tale penalizzazione di tipo ℓ_2 scoraggia rappresentazioni latenti di grande norma, favorendo codifiche più compatte e contribuendo alla stabilità del modello.

Un'alternativa è rappresentata dalla regolarizzazione di tipo ℓ_1 applicata allo spazio latente:

$$\arg \min_{f,g} \langle \Delta(\mathbf{x}_i, f(g(\mathbf{x}_i))) \rangle + \lambda \|\mathbf{h}_i\|_1. \quad (2.9)$$

A differenza della norma ℓ_2 , la regolarizzazione ℓ_1 tende a produrre rappresentazioni sparse, in cui solo un numero limitato di componenti del vettore latente risulta attivo per ciascun input. Questo comportamento favorisce una decomposizione più interpretabile delle feature e costituisce il principio alla base degli *Sparse Autoencoders*, che verranno analizzati nel seguito.

L'aggiunta di vincoli nella funzione obiettivo consente quindi di superare i limiti degli autoencoders classici, guidando l'apprendimento verso soluzioni non banali e semanticamente più significative, anche in assenza di una riduzione esplicita della dimensionalità dello spazio latente.

2.3.3 Relazioni con la PCA

Dal momento che gli autoencoders possono essere utilizzati per la riduzione della dimensionalità dei dati, è di interesse evidenziare la loro relazione con il metodo delle *Principal Component Analysis* (PCA). La PCA è una tecnica di analisi statistica che consente di ridurre la dimensionalità di un dataset preservando la maggior parte della varianza presente nei dati originali, mediante una trasformazione lineare delle variabili.

Sia dato un dataset di M osservazioni centrate

$$\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_M \in \mathbb{R}^n, \quad \frac{1}{M} \sum_{i=1}^M \mathbf{x}_i = 0.$$

Definendo la matrice dei dati

$$X = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_M^\top \end{bmatrix} \in \mathbb{R}^{M \times n},$$

la matrice di covarianza empirica è data da

$$C = \frac{1}{M} X^\top X \in \mathbb{R}^{n \times n}.$$

L'obiettivo della PCA consiste nell'individuare una direzione unitaria $\mathbf{w}_1 \in \mathbb{R}^n$ lungo la quale la proiezione dei dati presenti la massima varianza. Indicando con $y_i = \mathbf{w}_1^\top \mathbf{x}_i$ la proiezione dell'osservazione \mathbf{x}_i lungo tale direzione, la varianza dei dati proiettati può essere espressa come

$$\text{Var}(X \mathbf{w}_1) = \frac{1}{M} \sum_{i=1}^M (\mathbf{w}_1^\top \mathbf{x}_i)^2,$$

dove si è utilizzato il fatto che i dati sono centrati, e quindi la media delle proiezioni risulta nulla. Riscrivendo la precedente espressione in forma matriciale si ottiene

$$\text{Var}(X \mathbf{w}_1) = \mathbf{w}_1^\top \left(\frac{1}{M} \sum_{i=1}^M \mathbf{x}_i \mathbf{x}_i^\top \right) \mathbf{w}_1 = \mathbf{w}_1^\top C \mathbf{w}_1.$$

Il problema della ricerca della direzione di massima varianza può quindi essere formulato come il seguente problema di ottimizzazione vincolata:

$$\max_{\|\mathbf{w}_1\|_2=1} \mathbf{w}_1^\top C \mathbf{w}_1. \quad (2.10)$$

2.3. IL PROBLEMA DELL'IDENTITÀ E LA NECESSITÀ DI VINCOLI⁹



Figura 2.2: L'immagine mostra Davide Mononcello in uno stato di sonno

Tale problema può essere risolto mediante il metodo dei moltiplicatori di Lagrange, introducendo la lagrangiana

$$L(\mathbf{w}, \lambda) = \mathbf{w}^\top C \mathbf{w} - \lambda(\mathbf{w}^\top \mathbf{w} - 1).$$

Imponendo la condizione di stazionarietà rispetto a \mathbf{w} si ottiene

$$\nabla_{\mathbf{w}} L(\mathbf{w}, \lambda) = 2C\mathbf{w} - 2\lambda\mathbf{w} = 0,$$

da cui segue il problema agli autovalori

$$C\mathbf{w} = \lambda\mathbf{w}. \quad (2.11)$$

Le soluzioni ammissibili sono pertanto gli autovettori di C , mentre i moltiplicatori di Lagrange coincidono con i corrispondenti autovalori. La derivata della lagrangiana rispetto a λ restituisce inoltre il vincolo di normalizzazione

$$\mathbf{w}^\top \mathbf{w} = 1.$$

Sia \mathbf{v}_k un autovettore unitario di C associato all'autovalore λ_k . Per tali vettori vale

$$\text{Var}(X\mathbf{v}_k) = \mathbf{v}_k^\top C \mathbf{v}_k = \lambda_k,$$

ossia ciascun autovalore rappresenta la varianza dei dati lungo la corrispondente direzione \mathbf{v}_k .

Ordinando gli autovalori in ordine decrescente

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0,$$

le direzioni associate agli autovalori maggiori individuano le componenti principali del dataset. In particolare, la prima componente principale \mathbf{v}_1 è la direzione di massima varianza, mentre le componenti successive massimizzano la varianza residua sotto il vincolo di ortogonalità rispetto alle precedenti.

Caso di un autoencoder lineare

Si consideri un autoencoder costituito da un encoder e un decoder entrambi lineari, addestrato su dati centrati. L'encoder realizza una mappatura del tipo

$$\mathbf{h} = W\mathbf{x},$$

dove $W \in \mathbb{R}^{q \times n}$ e $q < n$ è la dimensione dello spazio latente. Il decoder ricostruisce l'input mediante

$$\hat{\mathbf{x}} = W^\top \mathbf{h} = W^\top W \mathbf{x},$$

dove, senza perdita di generalità, si è assunto che i pesi del decoder siano vincolati a essere la trasposta di quelli dell'encoder.

L'addestramento dell'autoencoder consiste nel minimizzare l'errore quadratico medio di ricostruzione:

$$\mathcal{L}(W) = \frac{1}{M} \sum_{i=1}^M \|\mathbf{x}_i - W^\top W \mathbf{x}_i\|_2^2. \quad (2.12)$$

Osservando che $W^\top W$ è una matrice simmetrica di rango al più q , tale termine può essere interpretato come una proiezione lineare sul sottospazio generato dalle righe di W . L'errore di ricostruzione misura quindi la distanza tra ciascun dato e la sua proiezione su tale sottospazio.

Sfruttando l'ipotesi di dati centrati, la funzione di costo può essere riscritta come

$$\mathcal{L}(W) = \frac{1}{M} \sum_{i=1}^M \|\mathbf{x}_i\|_2^2 - \frac{1}{M} \sum_{i=1}^M \|W \mathbf{x}_i\|_2^2, \quad (2.13)$$

dove il primo termine è indipendente da W . Ne consegue che minimizzare l'errore di ricostruzione equivale a massimizzare la quantità

$$\frac{1}{M} \sum_{i=1}^M \|W \mathbf{x}_i\|_2^2. \quad (2.14)$$

2.3. IL PROBLEMA DELL'IDENTITÀ E LA NECESSITÀ DI VINCOLI 11

Indicando con $\mathbf{w}_1, \dots, \mathbf{w}_q$ le righe di W , si ottiene

$$\frac{1}{M} \sum_{i=1}^M \|W \mathbf{x}_i\|_2^2 = \sum_{j=1}^q \frac{1}{M} \sum_{i=1}^M (\mathbf{w}_j^\top \mathbf{x}_i)^2 = \sum_{j=1}^q \text{Var}(X \mathbf{w}_j), \quad (2.15)$$

ossia la somma delle varianze dei dati proiettati lungo le direzioni \mathbf{w}_j .

Pertanto, il problema di addestramento dell'autoencoder lineare equivale alla ricerca di q direzioni ortonormali che massimizzino la varianza totale dei dati proiettati. Questo coincide esattamente con il problema risolto dalla PCA, la cui soluzione è fornita dagli autovettori della matrice di covarianza associati ai q maggiori autovalori.

In particolare, il minimo della funzione di costo è ottenuto quando le righe di W coincidono (a meno di una trasformazione ortogonale) con gli autovettori $\mathbf{v}_1, \dots, \mathbf{v}_q$ associati agli autovalori $\lambda_1 \geq \dots \geq \lambda_q$. In tal caso vale

$$W^\top W = P_q,$$

dove P_q denota il proiettore ortogonale sul sottospazio generato dalle prime q componenti principali.

Ne consegue che un autoencoder lineare, addestrato mediante minimizzazione dell'errore quadratico medio, apprende lo stesso sottospazio individuato dalla PCA. Le coordinate latenti possono differire da quelle ottenute tramite PCA per una trasformazione ortogonale, ma lo spazio latente appreso coincide con lo span delle prime q componenti principali, mostrando come la PCA possa essere interpretata come un caso particolare di autoencoder lineare.

Caso di un autoencoder non lineare

Si consideri ora un autoencoder in cui almeno uno tra encoder e decoder è una funzione non lineare. In particolare, si assuma un encoder del tipo

$$\mathbf{h} = g(\mathbf{x}) = \sigma(W\mathbf{x} + \mathbf{b}),$$

dove $\sigma(\cdot)$ è una funzione di attivazione non lineare applicata elemento per elemento, mentre il decoder ricostruisce l'input mediante una funzione generica

$$\hat{\mathbf{x}} = f(\mathbf{h}).$$

L'addestramento dell'autoencoder consiste ancora nella minimizzazione dell'errore quadratico medio di ricostruzione:

$$\mathcal{L}(f, g) = \frac{1}{M} \sum_{i=1}^M \|\mathbf{x}_i - f(g(\mathbf{x}_i))\|_2^2. \quad (2.16)$$

A differenza del caso lineare, la mappatura complessiva $\mathbf{x} \mapsto \hat{\mathbf{x}}$ non è più una proiezione lineare su un sottospazio di dimensione ridotta. Di conseguenza, la funzione di costo non può essere riscritta in termini di varianza proiettata, né ricondotta a un problema agli autovalori della matrice di covarianza. In particolare, non è più possibile esprimere l'errore di ricostruzione come differenza tra una quantità costante e la varianza dei dati proiettati lungo un insieme di direzioni fisse.

L'autoencoder non lineare è quindi in grado di catturare strutture complesse e non lineari presenti nel dataset, che non possono essere rappresentate in modo efficace mediante una combinazione lineare di componenti principali rendendo possibile l'apprendimento di rappresentazioni latenti più flessibili e adatte a dati che giacciono approssimativamente su varietà non lineari.

2.4 Interpretabilità delle feature latenti

Uno degli obiettivi centrali nell'apprendimento di rappresentazioni è ottenere codifiche latenti che non siano solamente utili per la ricostruzione dei dati, ma anche interpretabili dal punto di vista umano. Nel contesto degli autoencoders, tale interpretabilità è strettamente legata alla capacità del modello di catturare e separare i fattori di variazione che governano la generazione dei dati osservati.

Definizione (Fattori di variazione). Si definiscono *fattori di variazione* le variabili latenti, generalmente non osservabili, che parametrizzano il processo generativo dei dati e ne determinano le principali modalità di cambiamento. Ciascun fattore di variazione corrisponde a una dimensione semantica distinta secondo cui le osservazioni possono variare, come ad esempio la forma, la posizione, l'orientamento, il colore o la presenza di specifici oggetti. [3]

L'introduzione di una strozzatura nello spazio latente o di vincoli di regolarizzazione nella funzione obiettivo costringe l'autoencoder a comprimere l'infor-

mazione contenuta nei dati di input, preservando principalmente gli aspetti rilevanti ai fini della ricostruzione. In linea di principio, questo processo può favorire l'apprendimento di rappresentazioni latenti che riflettono i fattori di variazione sottostanti ai dati, anziché limitarsi a una memorizzazione non strutturata delle osservazioni.

In uno scenario ideale, le componenti dello spazio latente risultano semanticamente interpretabili: la variazione di una singola variabile latente corrisponde a una modifica controllata e riconoscibile di un attributo specifico dell'osservazione ricostruita. In tal caso, i valori quantitativi assunti dalle feature latenti possono essere ricondotti a descrizioni qualitative comprensibili, rendendo lo spazio latente non solo compatto, ma anche concettualmente significativo.

Tuttavia, nella pratica, l'interpretabilità delle feature latenti non è garantita. Gli autoencoders standard sono addestrati esclusivamente per minimizzare l'errore di ricostruzione e tendono pertanto a organizzare lo spazio latente in modo funzionale a tale obiettivo, senza alcuna esplicita pressione a separare o strutturare semanticamente l'informazione. Di conseguenza, le rappresentazioni apprese risultano spesso difficili da interpretare e caratterizzate da una forte mescolanza dei fattori di variazione.

2.4.1 Rappresentazioni latenti disentangled

Una rappresentazione latente si dice *disentangled* quando i diversi fattori di variazione che descrivono i dati sono codificati in componenti latenti distinte e, idealmente, statisticamente indipendenti. In una tale rappresentazione, ciascuna variabile latente controlla un singolo fattore di variazione, mentre risulta invariata rispetto agli altri.

In presenza di una rappresentazione disentangled, la manipolazione di una singola dimensione dello spazio latente produce una variazione interpretabile e localizzata nell'output ricostruito, senza influenzare gli altri attributi dell'osservazione. Questa proprietà rende le rappresentazioni disentangled particolarmente desiderabili in applicazioni quali l'analisi esplorativa dei dati, il controllo generativo, la robustezza a variazioni spurie e il trasferimento di conoscenza tra domini.

Nonostante il loro interesse teorico e pratico, le rappresentazioni disentangled non emergono spontaneamente nell'addestramento di autoencoders classici. La sola presenza di una strozzatura dimensionale non è sufficiente a garantire la separazione dei fattori di variazione, e in molti casi il modello apprende

combinazioni complesse e non interpretabili di tali fattori, dando luogo a rappresentazioni *entangled*.

Per favorire l'apprendimento di rappresentazioni disentangled è quindi necessario introdurre vincoli aggiuntivi o specifiche scelte architetturali e di regolarizzazione. Tra queste rientrano l'imposizione di sparsità nello spazio latente, la promozione dell'indipendenza statistica tra le feature, o l'introduzione di termini di penalizzazione che incoraggino una separazione esplicita dei fattori di variazione. Tali strategie costituiscono la base di numerosi modelli avanzati, tra cui gli *Sparse Autoencoders*, che verranno analizzati nel seguito.

2.5 Sparse Autoencoders

Una possibile strategia per favorire l'apprendimento di rappresentazioni latenti astratte, disentangled e interpretabili consiste nell'introdurre esplicitamente vincoli di sparsità sulle attivazioni dello spazio latente. I modelli che adottano questa impostazione prendono il nome di **Sparse Autoencoders**.

A differenza degli autoencoder classici con strozzatura (bottleneck), nei quali la capacità di rappresentazione è limitata riducendo la dimensionalità dello spazio latente, negli Sparse Autoencoders si abbandona tale vincolo architetturale a favore di un vincolo di sparsità sulle attivazioni. In questo caso, lo spazio latente può avere dimensione pari o superiore a quella dell'input, e l'encoder realizza una mappatura del tipo

$$g : \mathbb{R}^n \rightarrow \mathbb{R}^q, \quad q \geq n, \quad (2.17)$$

richiedendo tuttavia che, per ciascun input, solo una frazione limitata delle unità latenti risulti significativamente attiva.

L'idea centrale è che, pur disponendo di uno spazio latente ad alta dimensionalità, il modello sia costretto a rappresentare ogni osservazione utilizzando un numero ridotto di componenti. Questo comportamento induce una codifica selettiva, nella quale le singole unità latenti tendono a rispondere a pattern o attributi specifici dei dati, favorendo rappresentazioni più strutturate e potenzialmente interpretabili.

Formalmente, dati un encoder g_θ e un decoder f_ϕ , la funzione obiettivo di uno Sparse Autoencoder può essere espressa come

$$\mathcal{L}(f_\phi, g_\theta) = \langle \Delta(\mathbf{x}_i, f_\phi(g_\theta(\mathbf{x}_i))) \rangle + \lambda \mathcal{R}_{\text{sparse}}(g_\theta(\mathbf{x}_i)), \quad (2.18)$$

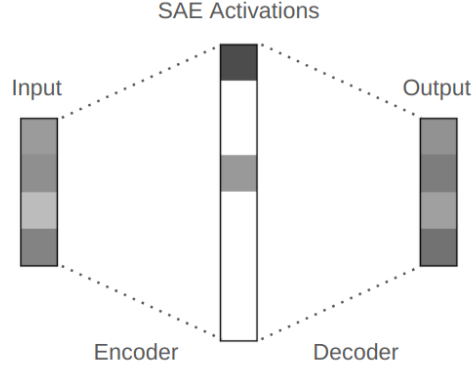


Figura 2.3: La figura mostra l'architettura di uno Sparse Autoencoder nel quale la dimensione dello stato latente è maggiore di quella di input.

dove $\mathcal{R}_{\text{sparse}}(\cdot)$ è un termine di regolarizzazione che impone vincoli di sparsità sulle attivazioni latenti.

Una delle scelte più comuni consiste nell'applicare una penalizzazione di tipo ℓ_1 alle attivazioni latenti,

$$\mathcal{R}_{\text{sparse}}(\mathbf{h}_i) = \|\mathbf{h}_i\|_1, \quad (2.19)$$

che incoraggia soluzioni in cui molte componenti del vettore latente sono nulle o prossime allo zero. In alternativa, è possibile imporre vincoli di tipo *top-k* (o *k-sparsity*), nei quali, per ciascun input, solo le k attivazioni di maggiore ampiezza vengono mantenute, mentre tutte le altre sono forzate a zero. Questo approccio impone una sparsità esplicita e controllata, indipendente dalla scala delle attivazioni.

Sebbene la sparsità non garantisca in senso rigoroso una completa separazione statistica dei fattori di variazione, essa introduce una forte pressione strutturale sulla rappresentazione latente, riducendo la codifica diffusa dell'informazione e favorendo l'emergere di feature più selettive e spesso *mono-semantiche*. Per questo motivo, gli Sparse Autoencoders costituiscono uno strumento particolarmente efficace per l'analisi e l'interpretazione di rappresentazioni dense apprese da modelli complessi, oltre a rappresentare una base concettuale naturale per lo studio del disentanglement.

Capitolo 3

Embeddings

Nets are for fish; once you get
the fish you can forget the net.
Words are for meaning; once
you get the meaning you can
forget the words.

Zhuangzi

3.1 Introduzione

Quando leggiamo un testo, noi esseri umani siamo dotati della capacità di coglierne un aspetto di significato. Questo implica che dietro alle parole che leggiamo si cela una rappresentazione semantica che vorremmo potenzialmente far apprendere anche alle macchine. Dal momento che le macchine parlano con la lingua dei numeri e, non come noi, con quella delle parole è stato necessario introdurre degli strumenti che vorrebbero in linea di principio assegnare ad ogni parola un numero rappresentativo di un significato. Tale strumenti sono chiamati *embeddings* e in questo capitolo, basandoci sul libro *Speech and Language Processing* di Stanford [4], si vedrà come vengono costruiti ed implementati per processare il testo.

3.2 L'ipotesi distribuzionale

Supponiamo di non conoscere il significato della parola *ongchoi*, ma di incontrarla nei seguenti contesti:

1. *L'ongchoi è deliziosa saltata con aglio.*
2. *L'ongchoi è ottima servita con riso.*
3. *...foglie di ongchoi con salse salate...*

Ora immaginiamo di aver già visto molte di queste parole-contesto in altri esempi, come:

1. *...gli spinaci saltati con aglio serviti sul riso...*
2. *...le coste, con i loro gambi e foglie, sono molto gustose...*
3. *...il cavolo riccio e altre verdure a foglia dal sapore salato...*

Il fatto che *ongchoi* compaia insieme a parole come *riso*, *aglio*, *deliziosa* e *salata*, proprio come *spinaci*, *coste* o *cavolo riccio*, suggerisce che l'*ongchoi* sia una **verdura a foglia** simile a queste altre verdure. Questo è il principio dell'ipotesi distribuzionale per il quale la parola *doctor-eye* o *oculist* è probabile che la troviamo nello stesso contesto.

Ipotesi Distribuzionale Si definisce ipotesi distribuzionale quella ipotesi per la quale parole simili compaiono in contesti simili.

Tale ipotesi suggerisce che il significato delle parole venga appreso sulla base del contesto di dove queste appaiono. Se questa intuizione viene seguita allora può divenire possibile trovare una soluzione per assegnare dei numeri a delle parole sulla base della loro occorrenza dentro contesti. Prima di arrivare però a capire come costruire gli embeddings è necessario introdurre una ulteriore intuizione attribuita ad Osgood nel 1957.

3.3 Ipotesi di Osgood

Un contributo fondamentale alla rappresentazione del significato proviene dal lavoro di Osgood et al. (1957), che studiarono la componente affettiva delle parole. Osgood mostrò che i giudizi emotivi associati a una parola possono essere descritti lungo tre dimensioni principali:

1. **Valenza**: quanto la parola è percepita come positiva o negativa.
2. **Arousal**: quanto la parola induce attivazione emotiva.
3. **Dominanza**: quanto la parola implica controllo o sottomissione.

Ogni parola può quindi essere rappresentata come una tripla di valori numerici che ne definiscono la posizione in questo spazio tridimensionale. Ad esempio:

$$heartbreak \rightarrow [2.5, 5.7, 3.6]$$

L'intuizione rivoluzionaria di Osgood è la seguente:

Ipotesi di Osgood

Il significato di una parola può essere rappresentato come un vettore in uno spazio semantico.

Questa idea è stata la prima ad anticipare direttamente i moderni modelli di *word embeddings*, in cui ogni parola è descritta come un punto in uno spazio multidimensionale corrispondente ad un significato.

3.4 Embeddings

L'unione dell'ipotesi distribuzionale e dell'ipotesi di Osgood ha aperto la strada agli embeddings come modello fondamentale per la rappresentazione computazionale del significato. Da un lato, l'ipotesi distribuzionale fornisce il principio secondo cui il significato delle parole può essere inferito dai contesti in cui esse compaiono; dall'altro, l'ipotesi di Osgood suggerisce che tale significato possa essere rappresentato come un vettore numerico in uno spazio semantico. In questa sezione introduciamo i primi modelli di embedding basati su conteggi, che costituiscono il punto di partenza storico e concettuale delle moderne rappresentazioni distribuzionali. Per orientare il lettore, è utile chiarire fin da subito le principali tipologie di embeddings che verranno introdotte nel seguito. In base alla natura della rappresentazione prodotta, è possibile distinguere due grandi famiglie: embeddings **statici** ed embeddings **dinamici**.

Embeddings statici Si definisce statico un embedding in cui ogni parola del vocabolario è associata a un unico vettore pre-computato. Tale rappresentazione rimane invariata a prescindere dal contesto specifico in cui la parola appare. (Esempi: Matrici di co-occorrenza, Word2Vec, GloVe).

Negli embeddings statici, a ciascun tipo di parola del vocabolario è associato un unico vettore, indipendente dal contesto in cui la parola appare. Questa categoria include sia gli embeddings distribuzionali basati su conteggi, come le matrici termine-documento e termine-termine eventualmente ridotte tramite SVD, sia gli embeddings predittivi appresi mediante modelli neurali, come Word2Vec. Gli embeddings dinamici, o contestuali, producono invece una rappresentazione dipendente dal contesto: la stessa parola può essere associata a vettori diversi a seconda della frase in cui compare. Tali rappresentazioni sono generate da modelli di linguaggio neurale profondi, a partire da architetture ricorrenti fino ai moderni modelli Transformer, come BERT.

Embeddings dinamici (Contextual) Si definisce dinamico un embedding in cui la rappresentazione vettoriale di una parola viene generata "al volo" in funzione dell'intera sequenza di input. La stessa parola riceve quindi vettori diversi a seconda del contesto semantico e sintattico circostante. (Esempi: ELMo, BERT, GPT).

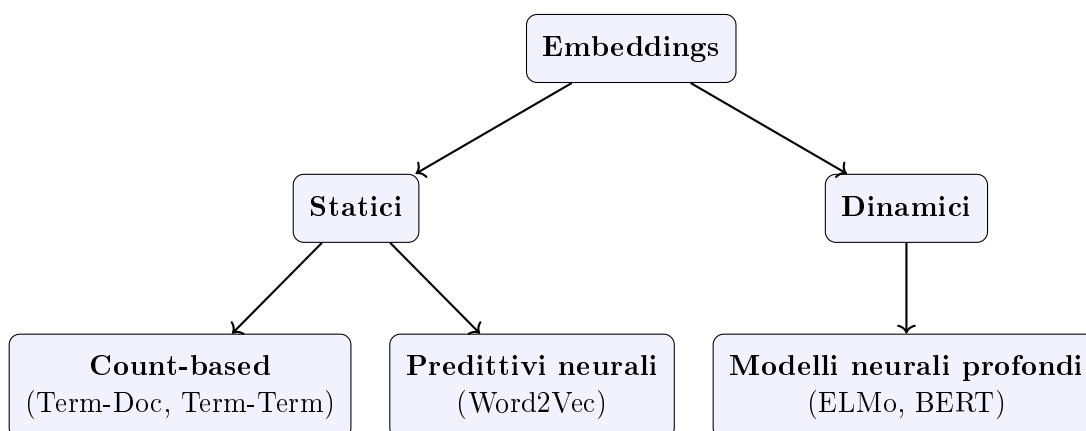


Figura 3.1: Classificazione delle principali tipologie di embeddings trattate nel capitolo.

3.4.1 Embeddings count-based

Il modo più semplice per costruire embeddings vettoriali delle parole è basato sulla **matrice di co-occorrenza**, una struttura che codifica quante volte determinati elementi linguistici compaiono insieme all'interno di un corpus. Esistono diverse varianti di matrici di co-occorrenza; in questa sezione ne introduciamo due fondamentali: la *term-document matrix* e la *term-term matrix*. Iniziamo dal caso più semplice.

Matrice termine-documento

In una matrice termine-documento ogni riga rappresenta una parola del vocabolario e ogni colonna rappresenta un documento appartenente a una collezione di testi. Ogni cella della matrice contiene il numero di volte in cui la parola associata alla riga compare nel documento associato alla colonna. Un esempio di term-document matrix è riportato nella Tabella 3.1, che mostra le occorrenze di quattro parole in quattro opere di Shakespeare.

Questa matrice può essere interpretata in due modi distinti ma complementari. Se si considerano le **colonne** della matrice, ciascun documento è rappresentato come un vettore in uno spazio di dimensione $|V|$, dove $|V|$ è la dimensione del vocabolario. In questo spazio, ogni asse corrisponde a una parola e il valore lungo ciascuna dimensione indica la frequenza della parola nel documento. Tale rappresentazione costituisce il fondamento del *vector space model* per il recupero dell'informazione, in cui documenti simili sono

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Tabella 3.1: Term-document matrix per quattro parole in quattro opere di Shakespeare. Ogni cella contiene il numero di occorrenze della parola (riga) nel documento (colonna). [3]

associati a vettori geometricamente vicini. Alternativamente, se si considerano le **righe** della matrice, ogni parola può essere interpretata come un vettore in uno spazio di dimensione pari al numero di documenti. In questo caso, le dimensioni del vettore non corrispondono più a parole, ma ai documenti del corpus, e il valore lungo ciascuna dimensione indica quanto frequentemente la parola compare in ciascun documento. Questa seconda interpretazione è particolarmente rilevante dal punto di vista semantico.

Interpretazione delle righe della matrice termine-documento

Due parole risultano simili se presentano distribuzioni simili sui documenti, ovvero se tendono a comparire negli stessi testi con frequenze comparabili.

La term-document matrix fornisce quindi una prima, semplice forma di *embedding distribuzionale* delle parole, in cui il significato emerge dalla loro distribuzione nei documenti del corpus.

Matrice termine-termine

Un'alternativa alla matrice termine-documento per la rappresentazione distribuzionale delle parole è la matrice termine-termine, detta anche *word-word matrix* o *term-context matrix*. In questo caso, le colonne della matrice non sono più etichettate da documenti, bensì da parole del vocabolario. La matrice ha quindi dimensionalità $|V| \times |V|$, dove $|V|$ indica la dimensione del vocabolario. In una matrice termine-termine, ogni riga rappresenta una **parola target** e ogni colonna rappresenta una **parola di contesto**. Ciascuna cella contiene il numero di volte in cui la parola di contesto compare nel contesto della parola target all'interno di un corpus di addestramento. Formalmente, la cella $M_{i,j}$ registra il numero di co-occorrenze tra la parola w_i

(target) e la parola w_j (contesto). Il concetto di *contesto* può essere definito in diversi modi. Una possibilità consiste nel considerare l'intero documento come contesto; tuttavia, nella pratica è molto più comune utilizzare contesti locali, definiti tramite una **finestra scorrevole** attorno alla parola target. Ad esempio, fissata una finestra di ampiezza $\pm k$, una parola è considerata di contesto se compare entro k posizioni a sinistra o a destra della parola target nel testo. Considerando tutte le occorrenze di ciascuna parola nel corpus e contando le parole che compaiono nelle rispettive finestre di contesto, è possibile costruire una matrice di co-occorrenza parola-parola. La Tabella 3.2 riporta un estratto reale di una matrice termine–termine calcolata sul corpus Wikipedia, in cui sono mostrate quattro parole target e alcune parole di contesto selezionate a scopo illustrativo [3].

Parola	aardvark	computer	data	result	pie	sugar
cherry	0	2	8	9	442	25
strawberry	0	0	0	1	60	19
digital	0	1670	1683	85	5	4
information	0	3325	3982	378	5	13

Tabella 3.2: Estratto di una matrice termine–termine calcolata sul corpus Wikipedia. Ogni cella contiene il numero di co-occorrenze tra la parola target (riga) e la parola di contesto (colonna) all'interno di una finestra di contesto locale [3].

In questa rappresentazione, ogni parola è associata a un vettore in uno spazio di dimensione $|V|$, in cui ciascuna dimensione corrisponde a una parola di contesto. Parole semanticamente simili tendono ad avere vettori simili, poiché compaiono in contesti linguistici analoghi. Ad esempio, dalla Tabella 3.2 si osserva che *cherry* e *strawberry* co-occorrono frequentemente con parole come *pie* e *sugar*, suggerendo una forte affinità semantica, mentre *digital* e *information* presentano distribuzioni simili rispetto a contesti come *computer* e *data*.

Interpretazione della matrice termine–termine

Due parole risultano semanticamente simili se presentano vettori di co-occorrenza simili, ovvero se tendono a comparire negli stessi contesti linguistici, anche nel caso in cui non compaiano mai direttamente insieme.

A questo punto abbiamo ottenuto una matrice termine–termine, le cui colonne sono le parole di contesto, e i valori le co-occorrenze. Data $|V|$ la dimensione

del vocabolario, tale matrice ha una dimensionalità

$$|V| \times |V|.$$

Si hanno tuttavia due problemi.

1. Dal momento che ogni parole co-ocorrerà solo con pochissime altre, *la dimensionalità della matrice è enorme.*
2. La maggior parte delle celle è nulla, e quindi *i vettori sono estremamente sparsi.*

Per affrontare i problemi legati all'elevata dimensionalità e alla natura estremamente sparsa della matrice termine-termine, ci sono diverse possibilità. Una di queste è la singular value decomposition in seugto descritta, e un'altra è quella di cambiare approccio e verrà presentata un'altra tipologia di embeddings basati su un altro paradigma di generazione diverso dal count-based che saranno successivamente presentati.

3.4.2 Riduzione dimensionale tramite SVD

Un metodo per la riduzione della dimensionalità è la Singular Value Decomposition applicata alla word-context matrix. Sia $M \in \mathbb{R}^{|V| \times |V|}$ la word-context matrix, eventualmente pesata tramite tf-idf. La decomposizione ai valori singolari (Singular Value Decomposition, SVD) consente di fattorizzare M come prodotto di tre matrici:

$$M = U \Sigma V^\top$$

dove U e V sono matrici ortogonali e Σ è una matrice diagonale contenente i valori singolari ordinati in modo decrescente. Ogni valore singolare rappresenta l'importanza di una direzione latente nello spazio semantico. I valori singolari maggiori catturano le correlazioni più rilevanti tra parole e contesti, mentre quelli più piccoli tendono a modellare rumore o variazioni locali meno informative. Per ottenere una rappresentazione a dimensionalità ridotta, si considera una versione troncata della decomposizione, mantenendo solo i primi k valori singolari:

$$M \approx U_k \Sigma_k V_k^\top$$

con $k \ll |V|$. Le righe della matrice $U_k \Sigma_k$ costituiscono una rappresentazione densa delle parole target in uno spazio latente di dimensione k . In

questo nuovo spazio, ogni parola è descritta da un vettore a dimensionalità ridotta, in cui le correlazioni semantiche risultano più evidenti rispetto alla rappresentazione originale sparsa. È importante osservare che la riduzione dimensionale non elimina esplicitamente la sparsità della matrice originale, ma proietta le parole in uno spazio denso in cui le relazioni semantiche emergono in forma compressa e più robusta.

3.4.3 Cosine Similarity

Dal momento che i vettori di embeddings vivono in uno spazio vettoriale che è anche uno spazio semantico, è possibile calcolare l'affinità di significato che due vettori hanno tramite la cosine similarity. La **cosine similarity** è una misura di similarità tra vettori che valuta il coseno dell'angolo compreso tra essi nello spazio vettoriale. Data la sua indipendenza dalla lunghezza dei vettori, risulta particolarmente adatta a confrontare vettori di frequenze o di pesi, come quelli derivati da matrici parola-contesto. Dati due vettori u e v , la similarità coseno è definita come:

$$\text{cosine_sim}(u, v) = \frac{u \cdot v}{\|u\| \|v\|} = \frac{\sum_i u_i v_i}{\sqrt{\sum_i u_i^2} \sqrt{\sum_i v_i^2}}. \quad (3.1)$$

Il valore risultante è compreso tra -1 e 1 :

- 1 indica che i vettori puntano nella stessa direzione (massima similarità),
- 0 indica che sono ortogonali (nessuna similarità),
- valori negativi indicano direzioni opposte (molto raro nei contesti di NLP).

Nelle applicazioni di elaborazione del linguaggio naturale la cosine similarity è spesso preferita alla distanza Euclidea, perché ci interessa confrontare il *pattern* delle co-occorrenze piuttosto che le loro magnitudini assolute. Ad esempio, due parole che co-occorrono con gli stessi termini di contesto, anche se con frequenze diverse, risulteranno comunque simili. La cosine similarity è quindi il principale strumento per valutare la similarità tra vettori distribuzionali e rappresenta un passaggio fondamentale prima di introdurre i modelli predittivi come Word2Vec e discendenti.

3.4.4 Word2Vec: un approccio predittivo

Sebbene i metodi basati su conteggi e la riduzione dimensionale tramite SVD permettano di ottenere rappresentazioni semanticamente dense, essi presentano limiti strutturali non trascurabili. Il calcolo della decomposizione ai valori singolari su matrici di co-occorrenza è computazionalmente oneroso, con una complessità che cresce sensibilmente rispetto alla dimensione del vocabolario, rendendo difficile la scalabilità su corpora massicci. Per superare queste criticità, Mikolov et al. (2013) hanno introdotto *Word2Vec*, un framework basato su un paradigma radicalmente diverso: la **predizione**. Invece di riassumere statistiche globali, Word2Vec apprende gli embeddings processando il testo localmente. Lo spostamento di paradigma risiede nel fatto che, anziché contare le occorrenze totali, addestriamo un classificatore su un compito di **classificazione binaria**. Il modello deve rispondere alla domanda:

“Data la parola target w (es. albicocca), qual è la probabilità che la parola candidata c (es. marmellata) compaia nel suo contesto?”

In questo approccio, noto come **self-supervision**, il testo stesso fornisce le etichette: ogni parola c che appare effettivamente vicino a w nel corpus fornisce un esempio positivo (etichetta 1). Al contrario, per addestrare il classificatore, il modello genera artificialmente degli esempi negativi campionando parole casuali dal vocabolario che non compaiono nel contesto di w .

Il classificatore e la funzione sigmoide

L'intuizione alla base del classificatore è che due parole siano semanticamente vicine se i loro vettori di embedding sono simili. Per misurare questa affinità, utilizziamo il **prodotto scalare** tra il vettore della parola target \mathbf{w} e il vettore della parola di contesto \mathbf{c} :

$$\text{Similarity}(w, c) \approx \mathbf{w} \cdot \mathbf{c}$$

Poiché il prodotto scalare può assumere qualsiasi valore reale, utilizziamo la funzione **sigmoide** $\sigma(x)$ per mappare il risultato in una probabilità compresa tra 0 e 1:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (3.2)$$

Il modello stima quindi la probabilità che la coppia (w, c) sia un esempio positivo (+) come:

$$P(+ \mid w, c) = \sigma(\mathbf{w} \cdot \mathbf{c}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{c}}} \quad (3.3)$$

Nel caso generale, data una parola target w e un'intera finestra di L parole di contesto $c_{1:L}$, il modello assume che le parole nel contesto siano indipendenti tra loro. La probabilità complessiva è dunque data dal prodotto delle probabilità individuali:

$$\log P(+ \mid w, c_{1:L}) = \sum_{i=1}^L \log \sigma(\mathbf{w} \cdot \mathbf{c}_i) \quad (3.4)$$

Apprendimento e Negative Sampling

L'addestramento consiste nello spostare iterativamente i vettori nello spazio semantico affinché il prodotto scalare tra parole che compaiono realmente insieme sia massimizzato, mentre quello tra parole prive di relazione sia minimizzato. Per rendere questo processo efficiente, si utilizza il **Negative Sampling**. Per ogni esempio positivo (w, c_{pos}) osservato nel testo, il modello genera k esempi negativi selezionando parole casuali dal vocabolario. Queste parole di “rumore” vengono scelte secondo una distribuzione unigramma pesata $P_\alpha(w)$:

$$P_\alpha(w) = \frac{\text{count}(w)^\alpha}{\sum_{w' \in V} \text{count}(w')^\alpha} \quad (3.5)$$

dove $\text{count}(w)$ indica la frequenza assoluta della parola w nel corpus, il simbolo w' al denominatore, che serve a calcolare il valore totale di normalizzazione, rappresenta l'indice della sommatoria che scorre su tutte le parole del vocabolario V e α è un parametro di smoothing (solitamente 0.75). Elevando le frequenze a questa potenza, si riduce la probabilità di campionare troppo spesso parole estremamente comuni e si aumenta quella delle parole più rare.

Perché due matrici? Il ruolo di W e C

Una caratteristica distintiva di Word2Vec è il mantenimento di **due rappresentazioni distinte** per ogni parola, organizzate in due matrici di pesi

separate (Figura 3.2). Una matrice W relativa alle parole target che contiene i vettori utilizzati quando la parola è il centro della finestra, e una matrice C che contiene i vettori utilizzati quando la parola appare nel contesto di un'altra o viene estratta come esempio negativo.

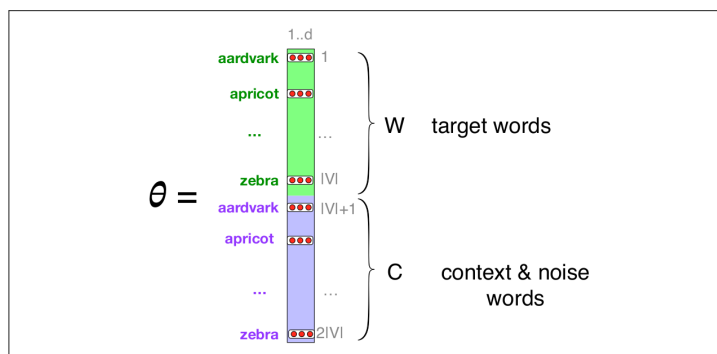


Figura 3.2: Lo skip-gram apprende in totale due insiemi di embedding, uno per i target (W) e uno per i contesti (C), per un totale di $2|V|$ vettori. L'addestramento mira a massimizzare la probabilità che parole vicine nel testo abbiano vettori simili.

Sdoppiando le matrici, il modello garantisce la **stabilità dell'ottimizzazione**. Se usassimo un unico vettore \mathbf{v} , il modello cercherebbe di massimizzare il prodotto scalare $\mathbf{v} \cdot \mathbf{v}$ (auto-similarità), portando i valori a crescere all'infinito. Con due matrici, il modello apprende relazioni distribuzionali senza questo vincolo. Al termine, si utilizzano solitamente i vettori di W o la media $W + C$.

3.4.5 Proprietà semantiche degli embeddings

L'apprendimento di questi vettori tramite il processo di ottimizzazione descritto non produce semplici sequenze numeriche prive di struttura, ma genera uno spazio geometrico capace di riflettere profonde relazioni linguistiche. La natura delle informazioni catturate da tali rappresentazioni dipende, in prima istanza, dalla configurazione della finestra di contesto, indicata come $L = 2m$, dove m rappresenta il raggio d'azione a destra e a sinistra della parola target. Una finestra ristretta (con m pari a 1 o 2) tende a privilegiare una similarità di tipo **sintattico**, raggruppando termini che condividono lo stesso ruolo grammaticale, come nel caso di verbi che occorrono in strutture frasali analoghe (ad esempio *scrive*, *dice* o *risponde*). Al contrario, l'adozione di una finestra più ampia (con m pari a 5 o più) sposta l'enfasi verso

una similarità di tipo **tematico** o tematico, associando parole che appartengono allo stesso ambito semantico, come *ospedale*, *ambulanza* e *infermiere*, indipendentemente dalla loro funzione sintattica immediata. Questa capacità di astrazione permette agli embeddings densi di catturare efficacemente la **similarità di secondo ordine**.

Associazione paradigmatica (Similarità di secondo ordine)

Due parole risultano vicine nello spazio vettoriale non perché compaiono necessariamente insieme nel testo (associazione sintagmatica di primo ordine), ma perché sono circondate da contesti simili.

Una delle manifestazioni più celebri di questa proprietà è la facoltà di supportare il ragionamento analogico attraverso il cosiddetto **modello del parallelogramma**. Poiché le relazioni semantiche sono codificate come differenze vettoriali costanti, è possibile risolvere analogie operando algebricamente sui vettori secondo la formula $b^* \approx b - a + a^*$. Tale regolarità permette di catturare relazioni grammaticali e semantiche complesse, dai passaggi di genere, come nella nota equazione $\text{king} - \text{man} + \text{woman} \approx \text{queen}$, fino ai rapporti geopolitici come $\text{Paris} - \text{France} + \text{Italy} \approx \text{Rome}$.

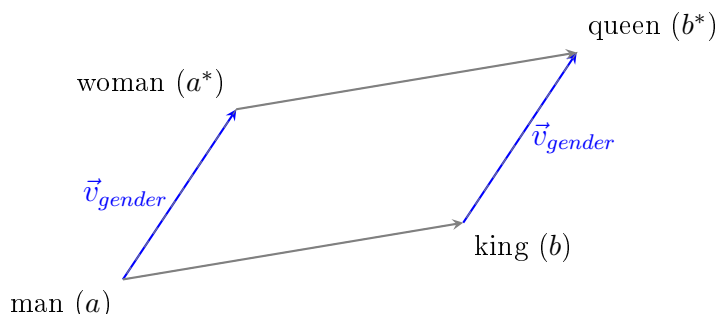


Figura 3.3: Rappresentazione geometrica del modello del parallelogramma applicato all'analogia di genere.

Da un punto di vista puramente geometrico, lo spazio degli embeddings è caratterizzato da proprietà di **parallelismo** e **ortogonalità**: mentre vettori paralleli indicano relazioni analoghe che si ripetono tra coppie diverse di parole, l'ortogonalità segnala l'indipendenza concettuale tra i termini. In sintesi, queste rappresentazioni apprese in modalità *self-supervised* incorporano automaticamente una vasta gamma di informazioni sintattiche e semantiche, rendendo la scelta della dimensione del vettore e dell'ampiezza della finestra i parametri critici per determinare la qualità del risultato finale. Tuttavia,

nonostante la loro straordinaria efficacia, questi modelli presentano un limite intrinseco dovuto alla loro natura **statica**: ogni parola del vocabolario è associata a un unico vettore che deve condensare in sé tutti i possibili sensi del termine, rendendo difficile la gestione della polisemia. Per superare questa rigidità e permettere alla rappresentazione di adattarsi dinamicamente al contesto specifico di ogni singola frase, la ricerca si è evoluta verso lo sviluppo degli **embeddings dinamici**.

3.5 Contextual Embeddings

Nelle sezioni precedenti abbiamo visto che gli embeddings statici, una volta completata la fase di addestramento, associano a ciascuna parola del vocabolario una rappresentazione vettoriale fissa. In questo approccio, il significato di una parola è modellato come una proprietà stabile e indipendente dal contesto in cui essa appare. Tuttavia, il significato linguistico non è un'entità immutabile, ma dipende in modo cruciale dal contesto sintattico e semantico in cui una parola viene utilizzata.

Una stessa parola può infatti esprimere significati diversi (polisemia), come nel caso di termini quali *spesso*, che può riferirsi sia a una frequenza sia a una caratteristica di spessore. Inoltre, anche quando il senso rimane invariato, il contributo semantico di una parola può variare in profondità e sfumature a seconda della frase in cui compare. Queste osservazioni rendono evidente il limite degli embeddings statici e motivano la necessità di rappresentazioni del significato capaci di adattarsi dinamicamente al contesto.

Per rispondere a questa esigenza sono stati introdotti i *contextual embeddings*, ovvero rappresentazioni vettoriali che modellano il significato di una parola come funzione dell'intera sequenza in cui essa appare. A differenza degli embeddings statici, gli embeddings contestuali non sono parametri direttamente associati alle parole del vocabolario, ma emergono come risultato dell'elaborazione di una sequenza di testo da parte di un modello di linguaggio neurale.

Dato un modello di linguaggio pre-addestrato e una nuova frase in input, è possibile interpretare la sequenza dei vettori di uscita del modello come un insieme di embeddings contestuali, uno per ciascun token della frase. Tali vettori catturano informazioni sintattiche e semantiche dipendenti dal contesto e possono essere utilizzati in qualsiasi task che richieda una rappresentazione del significato di parole o token in contesto.

Embeddings contestuali

Data una sequenza di token di input x_1, \dots, x_n , un modello di linguaggio produce una sequenza di vettori di uscita $h_1^{(L)}, \dots, h_n^{(L)}$, dove L denota l'ultimo strato del modello. Il vettore $h_i^{(L)}$ rappresenta il significato del token x_i nel contesto della sequenza x_1, \dots, x_n .

Nella pratica, anziché utilizzare esclusivamente il vettore di uscita dell'ultimo strato del modello, è comune costruire la rappresentazione contestuale di un token combinando le informazioni provenienti da più livelli. Una scelta frequente consiste nel calcolare la media dei vettori di uscita degli ultimi quattro strati, ovvero $h_i^{(L)}, h_i^{(L-1)}, h_i^{(L-2)}$ e $h_i^{(L-3)}$, ottenendo una rappresentazione più robusta che integra informazioni a diversi livelli di astrazione.

Questa distinzione è concettualmente fondamentale: mentre gli embeddings statici rappresentano il significato di *tipi di parola* (word types), gli embeddings contestuali rappresentano il significato di *istanze di parola* (word instances), ovvero di una parola specifica all'interno di un contesto specifico. In altre parole, il significato non è più una proprietà intrinseca della parola, ma il risultato della sua interazione con le altre parole della sequenza.

Per comprendere come tali rappresentazioni contestuali vengano effettivamente generate, è ora necessario analizzare il funzionamento dei modelli di linguaggio neurale, che costituiscono la sorgente primaria degli embeddings contestuali.

3.6 Modelli di linguaggio neurale

I *modelli di linguaggio neurale* costituiscono il meccanismo fondamentale attraverso cui vengono generate le rappresentazioni contestuali introdotte nella sezione precedente. L'idea centrale è che, per poter predire correttamente una parola in un determinato contesto, un modello debba necessariamente costruire una rappresentazione interna del contesto stesso. È proprio questa rappresentazione interna che, nei modelli moderni, viene interpretata come embedding contestuale.

Formalmente, un modello di linguaggio assegna una probabilità a una sequenza di token x_1, \dots, x_n . Nei modelli neurali tale probabilità viene fattorizzata tramite la regola della catena:

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i \mid x_1, \dots, x_{i-1}),$$

riducendo il problema alla stima della probabilità della parola successiva dato il contesto precedente. Il compito del modello è quindi quello di apprendere le regolarità statistiche, sintattiche e semantiche del linguaggio a partire da grandi corpora testuali.

Durante l'inferenza forward (o *decoding*), dato un contesto di parole precedenti, il modello esegue un forward pass sulla rete neurale per produrre una distribuzione di probabilità sulle possibili parole successive. Nei primi modelli di linguaggio neurale, il contesto osservabile è limitato a una finestra di dimensione fissa che considera un numero N di parole nel passato. Per chiarezza espositiva, consideriamo il caso $N = 3$, in cui il contesto è costituito dalle parole w_{t-1} , w_{t-2} e w_{t-3} . Ciascuna parola del contesto viene inizialmente rappresentata come un vettore *one-hot* di dimensione $|V|$, dove $|V|$ è la dimensione del vocabolario. Questi vettori vengono poi proiettati in uno spazio denso tramite una matrice di embedding $E \in \mathbb{R}^{d \times |V|}$. La moltiplicazione della matrice E per un vettore one-hot seleziona la colonna corrispondente alla parola, producendo il suo embedding vettoriale. Gli embedding delle parole di contesto vengono quindi concatenati per formare un unico vettore di input e .

Il vettore e viene successivamente trasformato da uno strato nascosto della rete, tramite una moltiplicazione per una matrice di pesi W , l'aggiunta di un termine di bias b e l'applicazione di una funzione di attivazione non lineare σ , ottenendo lo stato nascosto h . Lo stato nascosto viene quindi proiettato nello spazio delle dimensioni del vocabolario tramite una seconda matrice di pesi U , generando un vettore di punteggi z . L'ultimo passo consiste nell'applicazione della funzione *softmax*, che trasforma i punteggi in una distribuzione di probabilità. Dopo l'applicazione del softmax, ciascun nodo i dello strato di uscita stima la probabilità che la parola successiva w_t sia la parola del vocabolario con indice i , dato il contesto:

$$P(w_t = i \mid w_{t-1}, w_{t-2}, w_{t-3}).$$

In sintesi, le equazioni di un modello di linguaggio neurale feedforward con

finestra di contesto di dimensione 3, dati vettori di input one-hot per ciascuna parola di contesto, sono le seguenti:

$$\begin{aligned} e &= [Ex_{t-3}; Ex_{t-2}; Ex_{t-1}] \\ h &= \sigma(We + b) \\ z &= Uh \\ \hat{y} &= \text{softmax}(z) \end{aligned} \tag{3.1}$$

Si noti che il vettore di embedding e è ottenuto concatenando gli embedding delle parole di contesto; nel seguito utilizzeremo il punto e virgola per indicare la concatenazione di vettori. Sebbene questi modelli siano in grado di apprendere rappresentazioni distribuzionali utili e abbiano rappresentato un passo fondamentale verso la modellazione neurale del linguaggio, essi presentano limiti strutturali evidenti. In particolare, la dimensione del contesto è fissa e l'informazione contestuale viene compressa in un'unica rappresentazione, rendendo difficile catturare dipendenze a lungo raggio e strutture sintattiche complesse. Questi limiti hanno motivato lo sviluppo di modelli sequenziali più espressivi, in grado di aggiornare dinamicamente la rappresentazione del contesto man mano che la sequenza viene processata. Nel prossimo paragrafo analizzeremo le *Recurrent Neural Networks* (RNN) e le loro estensioni, le *Long Short-Term Memory* (LSTM), che rappresentano il primo tentativo sistematico di modellare il linguaggio come una sequenza di lunghezza variabile e di produrre embeddings contestuali dipendenti dall'intera storia precedente.

3.7 Reti neurali ricorrenti e LSTM

I modelli di linguaggio neurale feedforward introdotti nella sezione precedente rappresentano un primo passo verso la modellazione distribuzionale del linguaggio, ma soffrono di un limite strutturale fondamentale: il contesto è limitato a una finestra di dimensione fissa e non può adattarsi dinamicamente alla lunghezza della sequenza. Per superare questo vincolo, sono state introdotte le *Recurrent Neural Networks* (RNN), architetture progettate per elaborare sequenze di lunghezza arbitraria mantenendo una rappresentazione del contesto che viene aggiornata passo dopo passo.

In una RNN, la sequenza di input viene processata iterativamente. A ogni passo temporale t , il modello riceve in input il token corrente x_t e aggiorna uno *stato nascosto* h_t , che dipende sia dall'input corrente sia dallo stato

precedente:

$$h_t = f(h_{t-1}, x_t),$$

dove f è una funzione non lineare parametrizzata. Lo stato nascosto h_t costituisce una rappresentazione compatta del contesto osservato fino al passo t e può essere interpretato come una rappresentazione contestuale del token corrente.

Nei modelli di linguaggio basati su RNN, la probabilità della parola successiva viene stimata a partire dallo stato nascosto corrente:

$$P(w_t \mid w_1, \dots, w_{t-1}) = g(h_{t-1}),$$

dove g è tipicamente una trasformazione affine seguita da una funzione softmax. In questo modo, la RNN è in grado di modellare dipendenze sequenziali senza imporre un limite fisso alla dimensione del contesto, rappresentando un avanzamento significativo rispetto ai modelli feedforward.

Nonostante questi vantaggi, le RNN presentano importanti difficoltà nell'apprendere dipendenze a lungo raggio. Durante l'addestramento tramite backpropagation through time, i gradienti possono tendere a svanire o esplodere, rendendo difficile l'aggiornamento efficace dei parametri per sequenze lunghe. Questo problema, noto come *vanishing gradient problem*, limita la capacità delle RNN di catturare relazioni distanti nel testo.

Per affrontare tali limiti sono state introdotte le *Long Short-Term Memory* (LSTM), una variante delle RNN progettata per mantenere e aggiornare informazioni rilevanti su orizzonti temporali più lunghi. Le LSTM introducono una struttura di memoria esplicita, controllata da meccanismi di gating, che regolano quali informazioni debbano essere conservate, aggiornate o dimenticate nel tempo. Grazie a questi meccanismi, le LSTM risultano più stabili durante l'addestramento e più efficaci nel modellare dipendenze a lungo raggio.

Dal punto di vista delle rappresentazioni, gli stati nascosti prodotti da RNN e LSTM possono essere interpretati come le prime forme di *embeddings contestuali* appresi da modelli di linguaggio neurale. Ogni token è associato a un vettore che dipende dall'intera storia precedente della sequenza, consentendo di distinguere occorrenze diverse della stessa parola in contesti diversi.

Tuttavia, anche le LSTM presentano limiti strutturali rilevanti. In primo luogo, il contesto viene comunque compresso in un singolo stato nascosto,

che deve riassumere tutta l'informazione rilevante della sequenza precedente. In secondo luogo, le RNN e le LSTM elaborano la sequenza in modo intrinsecamente sequenziale, impedendo una parallelizzazione efficiente del calcolo. Infine, nei modelli di linguaggio standard, il contesto utilizzato è tipicamente unidirezionale, limitato alle parole precedenti.

Questi limiti hanno motivato lo sviluppo di modelli in grado di sfruttare informazioni provenienti sia dal contesto sinistro sia da quello destro di una parola. Nel prossimo paragrafo analizzeremo ELMo, un modello che introduce embeddings contestuali bidirezionali basati su reti ricorrenti, rappresentando un'importante tappa intermedia nel percorso che conduce ai modelli basati sull'architettura Transformer.

3.8 ELMo e il contesto bidirezionale

Le RNN e le LSTM introducono per la prima volta rappresentazioni contestuali dipendenti dalla sequenza, ma nei modelli di linguaggio standard tali rappresentazioni sono tipicamente *unidirezionali*, poiché il contesto utilizzato per predire una parola è limitato alle parole precedenti. Tuttavia, molti compiti di comprensione del linguaggio naturale richiedono di interpretare una parola alla luce dell'intera frase, includendo anche il contesto destro. Questa osservazione ha motivato lo sviluppo di modelli in grado di produrre rappresentazioni contestuali *bidirezionali*.

ELMo (*Embeddings from Language Models*) rappresenta uno dei primi modelli in grado di fornire embeddings contestuali profondi e bidirezionali. Il modello è basato su una architettura *biLSTM*, ovvero due LSTM separate che processano la sequenza in direzione sinistra-destra e destra-sinistra. Le rappresentazioni prodotte da entrambe le direzioni vengono quindi combinate per ottenere un embedding contestuale per ciascun token.

Un aspetto distintivo di ELMo è l'utilizzo di rappresentazioni *stratificate*. Il modello produce infatti più livelli di rappresentazione per ogni token, ciascuno dei quali cattura informazioni linguistiche a diversi livelli di astrazione. Le rappresentazioni finali utilizzate nei task downstream non corrispondono necessariamente all'output di un singolo strato, ma vengono ottenute come combinazione pesata degli stati prodotti dai diversi strati del modello.

Dal punto di vista semantico, ELMo fornisce embeddings distinti per ogni occorrenza di una parola, consentendo di catturare in modo efficace fenomeni

di polisemia e disambiguazione del senso. In questo senso, ELMo rappresenta un passaggio cruciale nel superamento definitivo del paradigma degli embeddings statici e dimostra empiricamente l'utilità delle rappresentazioni contestuali profonde per una vasta gamma di compiti linguistici.

Nonostante questi progressi, ELMo eredita alcuni limiti strutturali dalle architetture ricorrenti su cui è basato. In particolare, l'elaborazione della sequenza rimane intrinsecamente sequenziale, limitando la possibilità di parallelizzazione del calcolo. Inoltre, sebbene la bidirezionalità consenta di sfruttare l'intero contesto, l'informazione deve comunque essere mediata attraverso stati ricorrenti, rendendo difficile modellare direttamente dipendenze molto distanti nella sequenza.

Questi limiti hanno motivato l'introduzione di un nuovo paradigma architetturale, in cui le parole di una sequenza possono interagire direttamente tra loro senza essere mediate da uno stato ricorrente. Nel prossimo paragrafo introdurremo il meccanismo di *attention* e, in particolare, la *self-attention*, che costituisce il fondamento dei modelli Transformer.

3.9 Attention e Self-Attention

I modelli basati su RNN e LSTM, inclusi quelli bidirezionali come ELMo, producono rappresentazioni contestuali efficaci, ma presentano un limite strutturale comune: l'informazione contestuale deve essere mediata attraverso uno stato ricorrente che riassume l'intera sequenza. Questo meccanismo di compressione rende difficile modellare in modo diretto dipendenze a lungo raggio e relazioni complesse tra parole distanti nella frase.

Per superare questo limite è stato introdotto il meccanismo di *attention*, che permette a un modello di selezionare dinamicamente le parti più rilevanti del contesto quando deve produrre una rappresentazione o una predizione. L'idea fondamentale dell'*attention* è che, anziché affidarsi a una singola rappresentazione globale del contesto, il modello possa accedere direttamente a tutte le rappresentazioni disponibili e assegnare loro un peso in base alla rilevanza rispetto a un determinato obiettivo.

In termini intuitivi, il meccanismo di *attention* consente al modello di rispondere alla domanda: *quali parole del contesto sono più informative per interpretare il token corrente?* I pesi di attenzione determinano l'importanza

relativa di ciascun token del contesto e vengono utilizzati per combinare le rappresentazioni disponibili in una nuova rappresentazione contestuale.

Un'evoluzione fondamentale di questo meccanismo è rappresentata dalla *self-attention*. A differenza dell'attention classica, in cui l'attenzione è calcolata tra due sequenze distinte (ad esempio una sequenza sorgente e una sequenza target), nella self-attention ciascun token di una sequenza può attendere direttamente a tutti gli altri token della stessa sequenza. In questo modo, ogni parola costruisce la propria rappresentazione contestuale come combinazione pesata delle rappresentazioni di tutte le altre parole della frase.

La self-attention presenta diversi vantaggi rispetto ai modelli ricorrenti. In primo luogo, consente di modellare direttamente dipendenze a lungo raggio, poiché la distanza tra due token nella sequenza non influisce sulla loro capacità di interagire. In secondo luogo, l'elaborazione della sequenza non è più intrinsecamente sequenziale: le rappresentazioni di tutti i token possono essere calcolate in parallelo, migliorando significativamente l'efficienza computazionale e la scalabilità del modello.

Dal punto di vista delle rappresentazioni, la self-attention produce per ciascun token una rappresentazione contestuale che integra informazioni provenienti dall'intera sequenza. Il significato di una parola emerge quindi come risultato esplicito delle interazioni con tutte le altre parole della frase, piuttosto che come un riassunto implicito codificato in uno stato ricorrente.

Tuttavia, il meccanismo di attention da solo non definisce un modello completo di linguaggio o di rappresentazione. Per ottenere un'architettura in grado di produrre rappresentazioni contestuali profonde e composizionali è necessario combinare la self-attention con ulteriori componenti strutturali, come trasformazioni non lineari, meccanismi di normalizzazione e informazioni sulla posizione dei token nella sequenza.

Nel prossimo paragrafo introdurremo l'architettura Transformer, che integra il meccanismo di self-attention in una struttura modulare e profonda, costituendo il fondamento dei moderni modelli di linguaggio basati su embeddings contestuali, incluso BERT.

3.10 BERT e Masked Language Modeling

L'architettura Transformer encoder introdotta nella sezione precedente fornisce un meccanismo potente per la costruzione di rappresentazioni contestuali, ma da sola non determina come tali rappresentazioni debbano essere apprese né quale obiettivo di addestramento sia più adatto ai compiti di comprensione del linguaggio naturale. BERT (*Bidirectional Encoder Representations from Transformers*) rappresenta una risposta a questa esigenza, combinando il Transformer encoder con un obiettivo di addestramento specificamente progettato per produrre embeddings contestuali profondamente bidirezionali.

A differenza dei modelli di linguaggio causali, che predicono la parola successiva utilizzando esclusivamente il contesto sinistro, BERT utilizza un Transformer *encoder-only* e adotta un obiettivo di addestramento che consente al modello di sfruttare simultaneamente il contesto sinistro e destro di ciascun token. Questa caratteristica rende BERT particolarmente adatto a compiti interpretativi, come la classificazione di testo, il riconoscimento di entità nominate e la disambiguazione del senso delle parole.

Il principale obiettivo di addestramento di BERT è il *Masked Language Modeling* (MLM). Durante il pretraining, una frazione dei token di una frase viene mascherata e il modello è addestrato a predire i token originali a partire dal contesto circostante. In questo modo, il modello è costretto a costruire rappresentazioni che integrano informazioni provenienti da entrambe le direzioni della sequenza, dando luogo a embeddings contestuali profondamente bidirezionali.

Dal punto di vista architetturale, l'input di BERT è costituito dalla somma di tre componenti: l'embedding del token (tipicamente ottenuto tramite una tokenizzazione a sotto-parole), l'embedding di posizione e un embedding di segmento utilizzato per distinguere parti diverse dell'input. L'output del modello è una sequenza di vettori contestuali, uno per ciascun token, prodotti dall'ultimo strato del Transformer encoder.

Un elemento distintivo di BERT è l'introduzione di un token speciale [CLS], inserito all'inizio della sequenza di input. Il vettore contestuale associato a questo token viene spesso utilizzato come rappresentazione dell'intera sequenza in compiti di classificazione. Parallelamente, i vettori associati agli altri token forniscono rappresentazioni contestuali a livello di parola, utilizzabili per compiti di annotazione sequenziale e analisi semantica.

Come nei modelli contestuali precedenti, anche in BERT è comune non utilizzare esclusivamente il vettore di uscita dell'ultimo strato, ma combinare le rappresentazioni provenienti da più livelli del modello. In particolare, la media o la somma dei vettori degli ultimi strati consente di integrare informazioni a diversi livelli di astrazione, rendendo gli embeddings più robusti e informativi.

Il paradigma di addestramento di BERT segue lo schema *pretrain–finetune*. Nella fase di pretraining, il modello apprende rappresentazioni generali del linguaggio a partire da grandi quantità di testo non annotato. Nella fase di finetuning, tali rappresentazioni vengono adattate a specifici compiti downstream mediante l'aggiunta di teste di classificazione leggere e un addestramento supervisionato su dataset più piccoli.

Dal punto di vista delle rappresentazioni, BERT produce embeddings contestuali profondi, stratificati e ad alta dimensionalità, che incorporano in modo entangled informazioni sintattiche, semantiche e pragmatiche. Questa ricchezza rappresentativa è una delle principali ragioni del successo empirico di BERT, ma rende al contempo complessa l'interpretazione delle singole dimensioni e delle strutture latenti degli embedding.

Nel prossimo paragrafo discuteremo perché tali rappresentazioni, pur essendo estremamente efficaci, beneficiano di tecniche di analisi e *disentanglement*, motivando l'utilizzo di metodi basati su autoencoder sparsi per l'interpretazione e la scomposizione degli embeddings di BERT, che costituisce l'obiettivo principale di questa tesi.

3.11 Motivazione per il disentanglement degli embeddings di BERT

Il percorso seguito in questo capitolo ha mostrato come gli embeddings moderni, in particolare quelli prodotti da BERT, rappresentino il punto di arrivo di una progressiva evoluzione delle rappresentazioni distribuzionali del linguaggio: da vettori statici associati a tipi di parola a rappresentazioni contestuali profonde, dipendenti dall'intera sequenza e apprese tramite modelli di linguaggio neurali bidirezionali.

Gli embeddings di BERT sono caratterizzati da un'elevata capacità rappresentativa. Essi catturano simultaneamente informazioni sintattiche, seman-

tiche e pragmatiche, distribuite su molte dimensioni e stratificate lungo i diversi livelli del modello. Questa ricchezza informativa è alla base delle eccellenti prestazioni empiriche di BERT in una vasta gamma di compiti di elaborazione del linguaggio naturale.

Tuttavia, tale potenza rappresentativa ha un costo in termini di interpretabilità. Le informazioni codificate negli embeddings di BERT risultano fortemente *entangled*: singole dimensioni non corrispondono a proprietà linguistiche chiaramente interpretabili e le strutture latenti che emergono nello spazio vettoriale sono difficili da analizzare direttamente. Di conseguenza, comprendere quali fattori semantici o sintattici contribuiscano a una determinata rappresentazione diventa un compito non banale.

Questo problema è particolarmente rilevante nel contesto di applicazioni che richiedono trasparenza, analisi qualitativa o controllo delle rappresentazioni interne del modello. In tali scenari, non è sufficiente disporre di embeddings accurati: è necessario poterli interpretare, analizzare e, in alcuni casi, scomporre in componenti latenti più semplici e semanticamente coerenti.

Le tecniche di *disentanglement* delle rappresentazioni mirano proprio a questo obiettivo: separare i fattori latenti che contribuiscono alla costruzione di una rappresentazione densa, rendendo esplicite strutture che risultano altrimenti sovrapposte. In questo contesto, gli autoencoder sparsi rappresentano uno strumento particolarmente adatto, poiché consentono di apprendere rappresentazioni latenti compatte in cui solo un numero limitato di componenti è attivo per ciascun input.

Applicare tecniche di disentanglement agli embeddings di BERT consente quindi di analizzare la struttura interna di queste rappresentazioni, individuare dimensioni o fattori latenti interpretabili e studiare come diverse proprietà linguistiche emergano nello spazio vettoriale. Questo approccio permette di coniugare l'elevata capacità rappresentativa dei modelli di linguaggio moderni con un maggiore grado di interpretabilità.

Nel capitolo successivo introdurremo il formalismo degli autoencoder e, in particolare, degli *sparse autoencoders*, discutendone le proprietà teoriche e il loro utilizzo come strumento per il disentanglement di rappresentazioni dense. Successivamente, tali tecniche verranno applicate agli embeddings di BERT attraverso l'applicazione sviluppata in questa tesi, fornendo un caso di studio concreto sull'analisi e l'interpretazione delle rappresentazioni contestuali.

3.12 Pooling

Data una sequenza di token

$$(t_1, \dots, t_N),$$

ottenuta dalla tokenizzazione di un testo e fornita in input a un modello Transformer di tipo BERT-like, il modello non restituisce una singola rappresentazione vettoriale dell'intera sequenza, bensì una sequenza di rappresentazioni contestuali a livello di token. Più precisamente, l'output del modello può essere espresso come una matrice

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \vdots \\ \mathbf{h}_N \end{bmatrix} \in \mathbb{R}^{N \times d},$$

dove d indica la dimensionalità dello spazio latente del modello e ciascun vettore $\mathbf{h}_i \in \mathbb{R}^d$ rappresenta l'embedding contestuale del token t_i , ovvero una codifica che incorpora informazione proveniente dall'intera sequenza grazie al meccanismo di self-attention. In molte applicazioni di elaborazione del linguaggio naturale, tuttavia, non si è interessati a una rappresentazione a livello di singolo token, bensì a un unico vettore che riassume il significato complessivo dell'intera sequenza, come nel caso della similarità semantica, del clustering di testi o della classificazione a livello di frase o documento. Per ottenere tale rappresentazione globale è quindi necessario introdurre un'operazione di *pooling*.

Formalmente, un'operazione di pooling è una funzione che aggrega una collezione di vettori in un singolo vettore:

$$\text{Pooling} : \mathbb{R}^{N \times d} \rightarrow \mathbb{R}^d.$$

Applicando tale funzione alla matrice di embedding token-level \mathbf{H} , si ottiene un vettore

$$\mathbf{v} = \text{Pooling}(\mathbf{H}),$$

che può essere interpretato come una rappresentazione vettoriale globale della sequenza di input, intesa come unità semantica. L'operazione di pooling realizza dunque una riduzione strutturale lungo la dimensione dei token, trasformando una rappresentazione distribuita su N elementi in un singolo punto nello spazio latente. La scelta della funzione di pooling non è univoca e dipende sia dal modello utilizzato sia dal tipo di informazione che si desidera preservare.

Strategie di pooling

Le principali strategie di pooling utilizzate per ottenere una rappresentazione vettoriale globale a partire dalle rappresentazioni token-level prodotte da modelli Transformer possono essere descritte come segue.

- **Mean pooling.** Una delle strategie più semplici e diffuse consiste nel calcolare la media aritmetica dei vettori associati ai singoli token:

$$\mathbf{v} = \frac{1}{N} \sum_{i=1}^N \mathbf{h}_i.$$

Questa operazione assegna a ciascun token lo stesso peso e produce una rappresentazione che riflette il contributo medio dell'intera sequenza. Il mean pooling è particolarmente adatto a compiti di similarità semantica e ad analisi distribuzionali del significato, poiché preserva l'informazione globale senza privilegiare posizioni specifiche della sequenza.

- **Max pooling.** Un'alternativa è rappresentata dal max pooling, in cui ciascuna componente del vettore finale è ottenuta selezionando il valore massimo osservato tra tutti i token:

$$\mathbf{v}_j = \max_{i=1, \dots, N} (\mathbf{h}_{i,j}),$$

dove $\mathbf{h}_{i,j}$ indica la j -esima componente del vettore \mathbf{h}_i . Questa strategia enfatizza le feature più salienti presenti nella sequenza, catturando segnali forti anche se localizzati in un numero limitato di token, ma tende a sacrificare l'informazione distribuita lungo l'intero testo.

- **Token speciale di classificazione ([CLS]).** Nei modelli BERT-like è comune utilizzare il vettore associato a un token speciale, tipicamente [CLS], come rappresentazione globale della sequenza:

$$\mathbf{v} = \mathbf{h}_{[\text{CLS}]}$$

In questo caso, il pooling non consiste in un'aggregazione esplicita dei vettori token-level, ma nella selezione di un singolo vettore. Il token [CLS] viene infatti addestrato durante il pretraining del modello a catturare informazione a livello di sequenza, svolgendo il ruolo di una rappresentazione globale implicita, particolarmente adatta a task di classificazione.

- **Pooling appreso.** Una classe più generale di approcci consiste nell'apprendere l'operazione di pooling tramite un modello parametrico. In questo caso, la rappresentazione globale è ottenuta applicando una funzione P_θ , tipicamente implementata come una rete neurale, alla matrice delle rappresentazioni token-level:

$$\mathbf{v} = P_\theta(\mathbf{H}).$$

Questo approccio consente di assegnare pesi differenti ai token e di modellare interazioni più complesse rispetto alle strategie statiche, al costo di una maggiore complessità del modello e della necessità di obiettivi di addestramento specifici.

Alcune famiglie di modelli, come i *Sentence-Transformers*, integrano direttamente l'operazione di pooling all'interno dell'architettura. In tali modelli, un backbone Transformer produce le rappresentazioni token-level, che vengono immediatamente aggregate tramite una delle strategie descritte (tipicamente mean pooling), e l'intero sistema è addestrato end-to-end per produrre embedding a livello di frase ottimizzati per compiti di similarità semantica e retrieval.

3.12.1 Pooling gerarchico per sequenze lunghe

I modelli Transformer di tipo BERT-like impongono un vincolo strutturale sulla lunghezza massima della sequenza di input, tipicamente indicata con L_{\max} . Quando una sequenza di token supera tale limite, non è possibile processarla integralmente in un singolo forward pass del modello. Sia quindi dato un testo tokenizzato come

$$T = (t_1, \dots, t_N),$$

con $N > L_{\max}$. In questo caso, per ottenere una rappresentazione vettoriale globale del testo senza ricorrere alla troncatura, è necessario adottare una strategia di aggregazione su più livelli.

Segmentazione in chunk La sequenza T viene suddivisa in K sotto-sequenze contigue (chunk)

$$C_1, C_2, \dots, C_K,$$

ciascuna di lunghezza al più $W_{\text{eff}} \leq L_{\text{max}}$, dove W_{eff} rappresenta la lunghezza massima effettivamente disponibile per i token di contenuto, una volta considerati eventuali token speciali richiesti dal modello. Formalmente,

$$C_k = (t_{(k-1)W_{\text{eff}}+1}, \dots, t_{\min(kW_{\text{eff}}, N)}), \quad k = 1, \dots, K.$$

Pooling a livello di token (intra-chunk) Ciascun chunk C_k viene processato indipendentemente dal modello Transformer, che produce una matrice di rappresentazioni token-level

$$\mathbf{H}^{(k)} \in \mathbb{R}^{m_k \times d},$$

dove m_k è il numero di token del chunk k e d è la dimensionalità latente del modello. Applicando una funzione di pooling a livello di token, si ottiene un embedding rappresentativo del singolo chunk:

$$\mathbf{e}_k = \text{Pooling}_{\text{token}}(\mathbf{H}^{(k)}), \quad \mathbf{e}_k \in \mathbb{R}^d.$$

Questa prima operazione di pooling realizza una riduzione dalla rappresentazione distribuita sui token a una rappresentazione compatta a livello di segmento.

Pooling a livello di chunk (inter-chunk) Una volta ottenuti gli embedding dei singoli chunk $\mathbf{e}_1, \dots, \mathbf{e}_K$, è necessario aggregarli per costruire una rappresentazione globale dell'intero testo. Tale aggregazione può essere formalizzata come una seconda operazione di pooling:

$$\mathbf{v}_D = \text{Pooling}_{\text{chunk}}(\mathbf{e}_1, \dots, \mathbf{e}_K), \quad \mathbf{v}_D \in \mathbb{R}^d.$$

Nel caso più semplice, adottato in questo lavoro, la funzione $\text{Pooling}_{\text{chunk}}$ coincide con la media aritmetica:

$$\mathbf{v}_D = \frac{1}{K} \sum_{k=1}^K \mathbf{e}_k.$$

Interpretazione come pooling gerarchico La procedura descritta può essere interpretata come una forma di *pooling gerarchico*. Il primo livello di pooling aggrega l'informazione a livello di token all'interno di ciascun chunk, mentre il secondo livello aggrega l'informazione a livello di chunk sull'intero documento. In questo modo, è possibile ottenere una rappresentazione vettoriale globale anche per testi di lunghezza arbitraria, preservando il contributo semantico di tutte le parti della sequenza e mantenendo la compatibilità con i vincoli architetturali del modello di embedding.

Capitolo 4

Disentangling Dense Embeddings with Sparse Autoencoders

4.1 Motivazione e contesto

I modelli di linguaggio basati su architetture Transformer producono embeddings contestuali estremamente ricchi ed efficaci dal punto di vista empirico. Tuttavia, queste rappresentazioni collocano il testo in spazi vettoriali ad alta dimensionalità le cui dimensioni non risultano direttamente interpretabili dal punto di vista semantico. Il significato è codificato in modo distribuito e fortemente entangled, rendendo difficile analizzare, spiegare o controllare le rappresentazioni interne del modello.

Questa mancanza di interpretabilità rappresenta un limite rilevante, in particolare in applicazioni che richiedono trasparenza, analisi qualitativa o manipolazione controllata del significato. Ne deriva l'esigenza di trasformare embeddings densi in rappresentazioni latenti più semplici e strutturate, in cui le componenti corrispondano a fattori semantici coerenti e, per quanto possibile, indipendenti. Questo obiettivo è comunemente indicato come *disentanglement* delle rappresentazioni.

In questo capitolo analizziamo il metodo proposto nel lavoro *Disentangling Dense Embeddings with Sparse Autoencoders* [3], che affronta il problema del disentanglement applicando sparse autoencoders agli embeddings prodotti da modelli di linguaggio pre-addestrati. L'idea centrale è che l'introduzione di vincoli di sparsità nello spazio latente favorisca l'emergere di feature interpretabili, consentendo di scomporre rappresentazioni dense ed entangled

in componenti semanticamente significative. Di seguito viene illustrata la metodologia proposta.

4.2 Metodologia e Architettura

Per ottenere il disentanglement degli embeddings densi, la metodologia adottata si basa sull'utilizzo di *Sparse Autoencoders* (SAE). A differenza degli autoencoder tradizionali, che comprimono l'input in uno spazio latente di dimensione inferiore (*bottleneck*), i SAE proiettano l'input in uno spazio latente sovradimensionato ($n \gg d$), imponendo tuttavia un forte vincolo di sparsità sulle attivazioni. Questa scelta architetturale consente di rappresentare ciascun embedding come combinazione lineare di un numero limitato di feature latenti, favorendo l'emergere di componenti semanticamente interpretabili e riducendo il fenomeno di sovrapposizione delle informazioni.

4.2.1 Definizione del Modello

Sia $x \in \mathbb{R}^d$ un vettore di embedding in input, prodotto da un modello di linguaggio pre-addestrato, e sia $h \in \mathbb{R}^n$ la rappresentazione latente, dove n indica il numero totale di feature latenti apprese dal modello. In pratica, n viene scelto come multiplo della dimensionalità originale d , così da ottenere una base latente sovracompleta.

L'architettura del SAE è composta da un encoder e un decoder, definiti come segue:

$$\text{Encoder: } h = f_{\theta}(x) = \sigma(W_e x + b_e) \quad (4.1)$$

$$\text{Decoder: } \hat{x} = g_{\phi}(h) = W_d h + b_d \quad (4.2)$$

dove:

- $W_e \in \mathbb{R}^{n \times d}$ e $W_d \in \mathbb{R}^{d \times n}$ sono rispettivamente le matrici dei pesi di codifica e decodifica;
- $b_e \in \mathbb{R}^n$ e $b_d \in \mathbb{R}^d$ sono i vettori di bias;
- $\sigma(\cdot)$ è una funzione di attivazione non lineare, tipicamente una ReLU.

Ogni colonna della matrice di decoder W_d può essere interpretata come una *feature latente*, ovvero una direzione nello spazio degli embedding che corrisponde a un concetto semantico appreso dal modello. La ricostruzione di un embedding avviene quindi come combinazione lineare di un piccolo sottoinsieme di queste direzioni.

4.2.2 Vincolo di Sparsità *k-Sparse*

Un elemento cruciale della metodologia è il meccanismo con cui viene imposta la sparsità. Invece di adottare una regolarizzazione L_1 standard, che può introdurre effetti indesiderati di *shrinkage* (riduzione sistematica dell'ampiezza delle attivazioni), viene utilizzato un vincolo *Top-k*.

Per ciascun input x , solo le k attivazioni più elevate nel vettore latente h vengono mantenute, mentre tutte le altre sono poste a zero. Questo garantisce che ogni embedding venga rappresentato attraverso un numero fisso e limitato di feature attive, rendendo la rappresentazione più interpretabile e favorendo un disentanglement più netto dei fattori semantici.

4.2.3 Funzione di Costo e Addestramento

L'obiettivo dell'addestramento del SAE è minimizzare l'errore di ricostruzione preservando al contempo la sparsità delle attivazioni. La funzione di perdita globale è definita come:

$$\mathcal{L}(\theta, \phi) = \frac{1}{d} \|x - \hat{x}\|_2^2 + \lambda \mathcal{L}_{\text{sparse}}(h) + \alpha \mathcal{L}_{\text{aux}}(x, \hat{x}) \quad (4.3)$$

Oltre al termine di ricostruzione principale, viene introdotta una *auxiliary loss* (\mathcal{L}_{aux}), ispirata alla tecnica dei *ghost gradients*. Questa componente ha lo scopo di mitigare il problema dei *dead latents*, ovvero feature che non si attivano mai durante il processo di addestramento.

Dato l'errore di ricostruzione del modello principale $e = x - \hat{x}$, la perdita ausiliaria è calcolata modellando l'errore residuo tramite un sottoinsieme di latenti inermi:

$$\mathcal{L}_{\text{aux}}(x, \hat{x}) = \|e - \hat{e}\|_2^2 \quad (4.4)$$

dove \hat{e} rappresenta la ricostruzione ottenuta utilizzando esclusivamente tali latenti. Questo meccanismo forza il modello a riutilizzare feature altrimenti inattive, migliorando la copertura dello spazio semantico.

4.3 Interpretazione Automatizzata delle Feature

Una volta addestrato il SAE, è necessario assegnare un significato semantico alle feature latenti apprese. Poiché il numero di feature può raggiungere decine o centinaia di migliaia, un'annotazione manuale risulta impraticabile. Per questo motivo viene adottato un approccio di interpretazione automatizzata basato su Large Language Models (LLM), utilizzati come *Interpreter*.

Per una data feature i , il processo di etichettatura avviene nei seguenti passaggi:

1. **Selezione degli esempi:** vengono individuati i documenti che producono le attivazioni più elevate per la feature i (*max activating examples*), insieme a documenti che non attivano la feature.
2. **Generazione dell'interpretazione:** l'LLM analizza entrambi gli insiemi di testi e identifica il concetto comune presente nei testi attivi ma assente negli altri.
3. **Astrazione:** il concetto individuato viene sintetizzato in una breve etichetta testuale, che rappresenta l'interpretazione semantica della feature.

Questo processo consente di collegare le direzioni vettoriali astratte dello spazio latente a concetti comprensibili dall'uomo.

4.4 Feature Families e Struttura Gerarchica

Le feature apprese da un SAE non sono indipendenti, ma mostrano relazioni di co-occorrenza e organizzazione gerarchica. Per catturare tali relazioni viene introdotto il concetto di *Feature Families*, ovvero gruppi di feature semanticamente correlate.

Una famiglia di feature è composta da una feature *genitore*, caratterizzata da un'elevata densità di attivazione e associata a un concetto più astratto e generale, e da più feature *figlie*, più sparse e specializzate, che rappresentano sottocategorie o istanze più specifiche del concetto genitore. È importante distinguere questo fenomeno dal *feature splitting*, che descrive l'evoluzione delle feature al variare della capacità del modello e non relazioni interne a un singolo SAE.

4.5 Feature Families e Struttura Gerarchica

Le feature apprese da uno Sparse Autoencoder non sono indipendenti, ma mostrano strutture di co-occorrenza e relazioni gerarchiche che riflettono l'organizzazione latente dei concetti semantici nello spazio degli embedding. Per analizzare e strutturare tali relazioni, viene introdotto il concetto di *Feature Families*, ovvero insiemi di feature semanticamente correlate che rappresentano uno stesso concetto a diversi livelli di astrazione.

Una feature family è composta da una feature *genitore*, associata a un concetto più generale e astratto, e da un insieme di feature *figlie*, più sparse e specializzate, che catturano sottocategorie o istanze più specifiche del concetto genitore. È importante distinguere questa struttura gerarchica interna a un singolo SAE dal fenomeno del *feature splitting*, che descrive invece come una stessa direzione semantica possa frammentarsi al crescere della capacità del modello e non riguarda relazioni tra feature all'interno dello stesso autoencoder.

Criterio di identificazione delle feature genitore e figlie La distinzione tra feature genitore e feature figlie non è determinata a partire da considerazioni semantiche esplicite, ma emerge da una proprietà statistica osservata nei dati, ovvero la *densità di attivazione* delle feature. Per ciascuna feature i viene definita la frequenza di attivazione

$$f_i = \sum_k A_{ik}, \quad (4.5)$$

che misura il numero di esempi del dataset sui quali la feature risulta attiva. Intuitivamente, feature con valori elevati di f_i tendono ad attivarsi in un ampio numero di contesti e quindi a rappresentare concetti più generali,

mentre feature con densità inferiore si attivano in contesti più ristretti e catturano concetti più specifici.

Questa osservazione viene utilizzata per indurre una gerarchia: date due feature connesse nel grafo di co-occorrenza, la feature con densità di attivazione maggiore viene interpretata come *genitore*, mentre quella con densità minore come *figlia*. In questo modo, il ruolo gerarchico delle feature è determinato in modo automatico e riproducibile a partire dalle statistiche di attivazione, senza ricorrere a supervisione semantica.

4.5.1 Costruzione del Grafo di Co-occorrenza

Per identificare le feature families, viene costruito un grafo che cattura i pattern di co-attivazione delle feature nel dataset. Per ogni coppia di feature i e j vengono calcolate le seguenti metriche:

- **Matrice di co-occorrenza:**

$$C_{ij} = \sum_k A_{ik} A_{jk}, \quad (4.6)$$

dove $A_{ik} = 1$ se la feature i è attiva sull'esempio k , e 0 altrimenti. Questo termine misura quante volte le due feature si attivano congiuntamente.

- **Matrice di similarità delle attivazioni:**

$$D_{ij} = \sum_k B_{ik} B_{jk}, \quad (4.7)$$

dove B_{ik} rappresenta il valore scalare dell'attivazione latente. Questa metrica tiene conto non solo della presenza congiunta delle feature, ma anche dell'intensità delle loro attivazioni.

Poiché feature molto frequenti tendono a co-attivarsi con molte altre in modo spurio, la matrice di co-occorrenza viene normalizzata rispetto alla frequenza f_i della feature i :

$$C_{ij}^{\text{norm}} = \frac{C_{ij}}{f_i + \epsilon}, \quad (4.8)$$

ottenendo una misura asimmetrica che approssima la probabilità che la feature j sia attiva dato che la feature i è attiva.

4.5.2 Identificazione delle Feature Families

Sulla matrice di co-occorrenza normalizzata viene applicata una soglia τ per eliminare relazioni deboli o rumorose. Nel lavoro originale viene utilizzato $\tau = 0.1$, valore che rappresenta un compromesso empirico tra robustezza delle relazioni individuate e copertura dello spazio semantico.

A partire dalla matrice sogliata, l'identificazione delle feature families avviene attraverso i seguenti passaggi:

1. costruzione di un *Maximum Spanning Tree* (MST) sul grafo di co-occorrenza, al fine di preservare le relazioni più forti evitando cicli;
2. orientamento degli archi confrontando la densità di attivazione dei nodi connessi, dirigendo ciascun arco dalla feature più densa (genitore) verso quella meno densa (figlia);
3. identificazione delle famiglie tramite una ricerca in profondità (DFS) a partire dai nodi radice, ovvero feature prive di archi entranti;
4. rimozione iterativa delle feature genitore e ricostruzione dell'MST per far emergere famiglie più fini o parzialmente sovrapposte.

Questo procedimento consente di ricostruire una struttura gerarchica dello spazio semantico latente, nella quale concetti astratti emergono come nodi ad alta connettività che aggregano progressivamente sottocategorie più specifiche, riflettendo l'organizzazione interna degli embeddings densi appresi dal modello.

Capitolo 5

Prisma

5.1 Introduzione

Prisma è un'applicazione nata con l'obiettivo di rendere interpretabili gli *embeddings* testuali, generalizzando questo processo a qualsiasi insieme di documenti. Gli *embeddings*, pur essendo rappresentazioni ricche dal punto di vista semantico, tendono a condensare molteplici concetti in uno spazio denso e difficilmente esplorabile, rendendo complessa l'analisi del loro contenuto.

L'idea alla base di Prisma richiama il comportamento di un prisma ottico: così come un fascio di luce bianca viene scomposto nelle sue componenti cromatiche fondamentali, allo stesso modo un *embedding* semanticamente denso può essere “rifratto” nei concetti che lo compongono. Il software si propone di decomporre queste rappresentazioni compatte in dimensioni concettuali interpretabili, fornendo una visione strutturata del contenuto semantico. Un obiettivo centrale è l'accessibilità: Prisma mira ad abbassare la barriera d'ingresso per utenti non specialisti, fornendo un'interfaccia intuitiva per esplorare rappresentazioni semantiche complesse.

5.2 Architettura

L'applicazione è sviluppata in Python utilizzando il framework Django come nucleo centrale per la gestione della logica di business e dell'interfaccia web. Il sistema si appoggia a un database SQLite locale per la persistenza dei dati e interagisce con il framework Ollama per l'esecuzione dei modelli di linguaggio (LLM) necessari all'interpretazione.

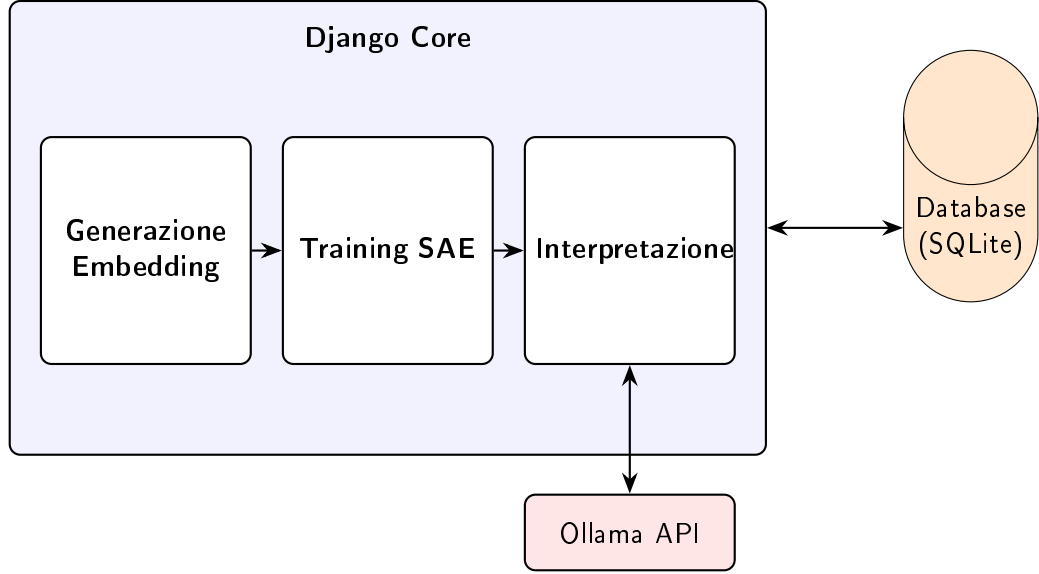


Figura 5.1: Schema semplificato dell'architettura di Prisma.

5.3 Generazione Embeddings

Il flusso operativo inizia con la creazione di un **Dataset** e la successiva trasformazione dei documenti in vettori numerici. L'applicazione permette di selezionare diversi modelli di codifica, tra cui modelli specializzati per il dominio clinico (come *medbit*) o modelli multilingua performanti (come la famiglia *GTE*).

5.3.1 Gestione di documenti lunghi: strategia *chunk-and-average*

Un aspetto strutturale del sistema Prisma riguarda la gestione di documenti testuali la cui lunghezza eccede il limite massimo di input dei modelli Transformer utilizzati per la generazione di embeddings. In particolare, molti modelli della famiglia BERT-like accettano in input una sequenza di lunghezza massima pari a

$$L_{\max} = 512 \text{ token.}$$

Di conseguenza, un documento con lunghezza superiore a L_{\max} non può essere processato integralmente in un singolo forward pass. Una soluzione naïve consiste nella troncatura (*truncation*) del testo, che mantiene solo i primi L_{\max} token; tuttavia, questa scelta può eliminare porzioni informative

rilevanti (ad esempio conclusioni, diagnosi o follow-up riportati in coda al documento). Per preservare l'intero contenuto informativo, Prisma implementa una strategia di segmentazione denominata *chunk-and-average*, che consente di codificare testi arbitrariamente lunghi producendo un singolo embedding globale per documento.

Tokenizzazione e vincoli di lunghezza Dato un documento testuale D , esso viene prima trasformato in una sequenza di token tramite un tokenizer associato al modello scelto. Indichiamo tale sequenza come

$$T = (t_1, t_2, \dots, t_N),$$

dove N è il numero totale di token ottenuti dalla tokenizzazione e i token t_i possono rappresentare parole intere o sotto-parole (subword), a seconda del vocabolario del modello. I modelli Transformer richiedono tipicamente l'aggiunta di token speciali che delimitano la sequenza e ne abilitano alcune funzionalità interne. Ad esempio, nei modelli tipo BERT è comune inserire un token di classificazione [CLS] all'inizio e un token di separazione [SEP] alla fine. Indichiamo con N_{special} il numero di token speciali necessari. La lunghezza massima effettivamente disponibile per il contenuto testuale (cioè per i token provenienti dal documento) è quindi

$$W_{\text{eff}} = L_{\text{max}} - N_{\text{special}}.$$

Segmentazione in chunk contigui Se $N \leq W_{\text{eff}}$, il documento può essere processato direttamente. Se invece $N > W_{\text{eff}}$, la sequenza T viene suddivisa in K segmenti contigui (*chunks*) ciascuno di lunghezza al più W_{eff} :

$$C_k = (t_{(k-1)W_{\text{eff}}+1}, \dots, t_{\min(kW_{\text{eff}}, N)}), \quad k = 1, \dots, K,$$

dove

$$K = \left\lceil \frac{N}{W_{\text{eff}}} \right\rceil.$$

Ciascun chunk viene poi confezionato come input valido per il modello aggiungendo i token speciali richiesti (ad esempio [CLS] e [SEP]), in modo che la lunghezza totale dell'input sia $\leq L_{\text{max}}$.

Embedding per chunk e concetto di pooling Sia f_θ il modello Transformer selezionato (con parametri θ). Dato un chunk C_k composto da m_k

token (dopo l'aggiunta dei token speciali), il forward pass del modello produce una sequenza di vettori nascosti (hidden states), uno per ciascun token:

$$f_\theta(C_k) = (\mathbf{h}_1^{(k)}, \mathbf{h}_2^{(k)}, \dots, \mathbf{h}_{m_k}^{(k)}), \quad \mathbf{h}_j^{(k)} \in \mathbb{R}^d,$$

dove d è la dimensionalità interna della rappresentazione (hidden size) del modello. Per ottenere un singolo vettore che rappresenti l'intero chunk, Prisma applica un'operazione di *pooling*. Con il termine pooling si intende una funzione di aggregazione che combina la sequenza di vettori token-level in un unico vettore a livello di segmento (segment-level). Nel caso più comune utilizzato in Prisma, si adotta il *mean pooling*, ovvero la media aritmetica dei vettori dei token (tipicamente escludendo eventuali token speciali, a seconda dell'implementazione):

$$\mathbf{e}_k = \text{MeanPool}(f_\theta(C_k)) = \frac{1}{m_k} \sum_{j=1}^{m_k} \mathbf{h}_j^{(k)}, \quad \mathbf{e}_k \in \mathbb{R}^d.$$

Intuitivamente, il mean pooling produce un vettore che riassume il contenuto semantico del chunk distribuendo il contributo su tutti i token che lo compongono, senza privilegiare una posizione specifica della sequenza.

Aggregazione a livello documento Una volta ottenuti gli embedding dei singoli chunk $\mathbf{e}_1, \dots, \mathbf{e}_K$, Prisma costruisce un embedding globale per il documento D combinandoli tramite media aritmetica:

$$\mathbf{v}_D = \frac{1}{K} \sum_{k=1}^K \mathbf{e}_k, \quad \mathbf{v}_D \in \mathbb{R}^d.$$

Questa scelta consente di ottenere una rappresentazione unica e confrontabile tra documenti, anche quando le loro lunghezze originali sono molto diverse. Inoltre, la procedura garantisce che tutte le parti del testo contribuiscano alla rappresentazione finale, evitando la perdita sistematica di informazione tipica della troncatura.

5.4 Training SAE

Una volta generati gli *embeddings*, il sistema procede alla loro scomposizione tramite il modulo *Sparse Autoencoders* (SAE). In questa fase, l'utente configura un'istanza di **SAERun** definendo parametri quali l'*expansion factor* (per

determinare la dimensione del livello latente) e la *k-sparsity* (per regolare il numero di neuroni attivi contemporaneamente).

Il training produce un modello capace di ricostruire l'input densificato attraverso una rappresentazione sparsa. Al termine, Prisma genera automaticamente analisi visive, tra cui mappe di calore della similarità delle matrici e della co-occorrenza, che permettono di valutare statisticamente la qualità delle feature estratte prima dell'interpretazione semantica.

5.5 Interpretazione

L'ultima fase del processo trasforma le feature matematiche in concetti comprensibili. Il modulo *Explorer* utilizza le attivazioni più significative di ogni feature per estrarre “evidenze” testuali dai documenti originali.

Queste evidenze vengono inviate all'API locale di Ollama, che genera un'etichetta (*label*) e una descrizione testuale per ogni feature. Prisma supporta la gestione di interpretazioni multiple per la stessa feature e permette di organizzare i concetti in **FeatureFamily**, ricostruendo una gerarchia semantica che va dai concetti generali a quelli più specifici.

Capitolo 6

Esperimenti e risultati

6.1 Introduzione

In questo lavoro di tesi sono stati condotti tre esperimenti principali. Il primo è stato chiamato Pedianet e riguarda dati medici che verranno in seguito presentati e descritti, il secondo basato su un dataset di abstracts di paper scientifici generale, il terzo invece riguarda un dataset di papers scientifici di PubMed. Verranno di seguito descritti e verranno presentati i risultati ottenuti.

6.2 Pedianet

Il progetto Pedianet è una rete italiana di pediatria di famiglia fondata nel 1998, con l'obiettivo di raccogliere, strutturare e analizzare i dati clinici pediatrici derivanti dalla normale attività ambulatoriale dei pediatri di libera scelta (PLS). L'iniziativa nasce per colmare una lacuna storica nella ricerca clinica pediatrica, ovvero la scarsa disponibilità di dati real-world relativi alla popolazione infantile. A differenza di altri database europei che includono anche adulti, Pedianet è uno dei pochi progetti esclusivamente focalizzati sull'età pediatrica, risultando così un archivio unico nel panorama della ricerca biomedica europea. La rete è composta da oltre 400 pediatri distribuiti su tutto il territorio italiano, che utilizzano la stessa piattaforma elettronica (Junior Bit) per la gestione delle cartelle cliniche dei pazienti. Circa 3700 pediatri utilizzano questo software, e sono potenziali candidati ad aderire alla rete. Tutti i dati raccolti vengono trasmessi tramite connessione protetta a

un server centrale situato a Padova, dove vengono validati, standardizzati e archiviati. I dati raccolti sono nella forma di cartelle cliniche pediatriche, ma presentano il problema di non essere strutturati, ovvero non organizzati e non separati in classi. Con la nascita dei large language models questi dati hanno assunto un valore completamente nuovo perché se correttamente lavorati e organizzati, dal momento che vanno a ricoprire un dominio molto specifico e verticale, si prestano benissimo a funzionare come dati di addestramento nel campo di modelli di NLP. Un'altra motivazione per cui è necessario intervenire con strumenti di NLP è quello di svolgere task di classificazione per destinare questi dati al mondo della ricerca. Per esempio si vorrebbe raccogliere tutte quelle cartelle che hanno visto casi di *bronchiolite*, ma non bastano metodi di regular expression dal momento che ci sono cartelle che invece presentano casi di *sospetta* bronchiolite. Si proverà quindi ad intervenire con la metodologia presentata nei precedenti capitoli per processare questi dati. Ecco il testo della sottosezione in formato paragrafo unico:

6.2.1 Scelta del modello di embedding

Per la generazione delle rappresentazioni vettoriali è stato selezionato il modello MedBIT [6], progettato per colmare il divario tecnologico nelle lingue meno fornite di risorse biomediche come l'italiano. Nel lavoro di riferimento, gli autori mettono a confronto due approcci distinti: BioBIT, basato sulla traduzione automatica neurale di milioni di abstract di PubMed per privilegiare la quantità dei dati, e MedBIT, che raffina tale conoscenza attraverso l'uso di un corpus di alta qualità composto da libri di testo medici scritti nativamente in italiano. La scelta di MedBIT per il progetto Pedianet risulta ottimale innanzitutto per la sua natività linguistica; poiché i dati di Pedianet sono redatti interamente in italiano dai medici di base, il modello garantisce una rappresentazione più naturale e sintatticamente corretta rispetto ai sistemi basati su traduzioni automatiche, riducendo le ambiguità semantiche. Queste caratteristiche rendono MedBIT lo strumento più affidabile per catturare le delicate sfumature cliniche presenti nei diari dei pediatri italiani, permettendo di distinguere efficacemente tra diagnosi accertate e semplici sospetti.

6.2.2 Esperimento

Per la conduzione dell'esperimento sono state processate circa 200.000 cartelle cliniche provenienti dal database Pedianet. Durante la fase preliminare di

analisi, è emersa una criticità significativa legata alla lunghezza dei testi: una porzione rilevante delle note cliniche superava il limite di lunghezza massima (L_{max}) imposto dall'architettura del modello MedBIT (tipicamente 512 tokens). La troncatura del testo (truncation) avrebbe comportato la perdita di informazioni potenzialmente cruciali situate nella parte finale delle note cliniche, compromettendo la qualità della rappresentazione semantica. Per ovviare a questo problema senza perdere contenuto informativo, è stata adottata una strategia di *chunk-and-average*. Formalmente, dato un documento D costituito da una sequenza di token $T = t_1, t_2, \dots, t_N$, dove $N > L_{max}$, il testo viene suddiviso in K segmenti (chunks) contigui C_1, C_2, \dots, C_K . La lunghezza effettiva utilizzabile per ogni chunk è definita come $W_{eff} = L_{max} - N_{special}$, dove $N_{special}$ rappresenta il numero di token speciali richiesti dal modello (es. [CLS] e [SEP]). Il documento viene quindi partizionato in modo tale che ogni chunk C_k contenga una sottosequenza di token di lunghezza al più W_{eff} :

$$C_k = t_i, \dots, t_{i+W_{eff}-1}$$

Per ogni segmento k , viene calcolato un embedding denso $\mathbf{e}_k \in \mathbb{R}^d$ attraverso il modello $f\theta$:

$$\mathbf{e}_k = \text{Pooling}(f\theta(C_k))$$

dove la funzione di pooling (nel nostro caso, *mean pooling*) aggrega gli stati nascosti dell'ultimo layer del modello per ottenere una singola rappresentazione vettoriale per il segmento. Infine, la rappresentazione vettoriale globale dell'intero documento clinico \mathbf{v}_D è ottenuta calcolando la media aritmetica degli embedding dei singoli segmenti. Questo approccio permette di condensare l'informazione distribuita lungo tutto il testo in un unico punto nello spazio latente:

$$\mathbf{v}_D = \frac{1}{K} \sum k = 1^K \mathbf{e}_k$$

Questa metodologia garantisce che ogni parte della cartella clinica contribuisca equamente alla rappresentazione finale, preservando dettagli sintomatologici o diagnostici che potrebbero trovarsi in qualsiasi punto della narrazione medica. A valle di questo processo di codifica, i vettori risultanti sono stati utilizzati per addestrare gli autoencoder sparsi (SAE) oggetto dello studio.

6.3 Abstracts

6.4 PubMed

Bibliografia

- [1] Dor Bank, Noam Koenigstein e Raja Giryes. “Autoencoders”. In: *CoRR* abs/2003.05991 (2020). arXiv: 2003.05991. URL: <https://arxiv.org/abs/2003.05991>.
- [2] Umberto Michelucci. *An Introduction to Autoencoders*. 2022. arXiv: 2201.03898 [cs.LG]. URL: <https://arxiv.org/abs/2201.03898>.
- [3] Xin Wang et al. *Disentangled Representation Learning*. 2024. arXiv: 2211.11695 [cs.LG]. URL: <https://arxiv.org/abs/2211.11695>.
- [4] Daniel Jurafsky e James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released August 24, 2025. 2025. URL: <https://web.stanford.edu/~jurafsky/slp3/>.
- [5] Mark Davies. *The Wikipedia Corpus: 4.6 million articles, 1.9 billion words*. <https://www.english-corpora.org/wiki/>. Adapted from Wikipedia. 2015.
- [6] Tommaso Mario Buonocore et al. “Localizing in-domain adaptation of transformer-based biomedical language models”. In: *Journal of Biomedical Informatics* 144 (2023), p. 104431. ISSN: 1532-0464. DOI: <https://doi.org/10.1016/j.jbi.2023.104431>. URL: <https://www.sciencedirect.com/science/article/pii/S1532046423001521>.