ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria
Laurea triennale in Ingegneria e Scienze Informatiche

INFORMATION RETRIEVAL BIOMEDICALE CON TESTO ARRICCHITO MEDIANTE KNOWLEDGE GRAPH CLINICO

Progetto in campo
Programmazione di Applicazioni Data Intensive

Relatore
Prof. Gianluca Moro

Co-relatore
Dott. Ing. Lorenzo Valgimigli

Presentato da
Elisa Barberini

Terza Sessione di laurea
Anno accademico 2022-2023

KEY WORDS

Natural Language Processing
Knowledge Graph
Deep Neural Networks
DPR
Deep Learning

Sommario

Un dominio dove il deep learning (DL), fondato su reti neurali con numerosi layer, ha trovato recentemente enorme successo, è quello del Natural Language Processing (NLP) ed una soluzione attualmente molto popolare è ChatGPT[1]. Il DL è di fatto impiegato con successo in tutti i task NLP[2], come machine translation, question answering, text summarization, sentiment analysis, information retrieval etc. Negli anni, numerose ricerche hanno evidenziato una relazione positiva tra il numero di parametri addestrabili ed i risultati ottenuti nella capacità di modellare sempre meglio il linguaggio naturale. Ciò ha spinto alla creazione dei Large Language Model (LLM)[3], modelli con centinaia di miliardi di parametri (Fig. 1). L'ampio numero di parametri permette a questi modelli di apprendere conoscenza generale sul linguaggio che viene successivamente sfruttata in tutti i task.

Un'area di ricerca innovativa è lo studio di metodi e tecniche per fare in modo che questi modelli possano sfruttare anche Knowledge Graph (KG), ossia grafi contenenti conoscenza su specifici domini. Questi KG, oltre a permettere una rappresentazione del dominio più accurata, potrebbero essere fondamentali per ridurre i dati di train necessari, rendere le deep neural network (DNN) più flessibili e veloci ad adattarsi a nuovi problemi.

L'obiettivo di questa tesi è duplice e consiste sia nello studio di un algoritmo per arricchire documenti biomedici utilizzando la conoscenza di KG, sia nell'addestramento di alcune DNN nella rappresentazione del linguaggio biomedico per il task di document retrieval.

In dettaglio proponiamo un algoritmo per arricchimento del testo biomedico, che sfrutta algoritmi State-Of-The-Art (SOTA) per la selezione dei termini biomedici più importanti (NER). Dopo di che, i termini selezionati verranno arricchiti dal punto di vista descrittivo e di relazioni con altre entità biomediche, sfruttando un Knowledge Graph clinico. In un secondo momento utilizziamo l'algoritmo proposto nel task di document retrieval dove per ogni query espressa in linguaggio naturale occorre selezionare i documenti più attinenti. Questo perché in letteratura è noto che una maggiore informazione espressa nella query porti ad una più precisa selezione dei documenti. Si vuole quindi verificare se determinati language model, specifici per il dominio biomedico, ottengano

performance migliori attraverso l'utilizzo del testo arricchito.

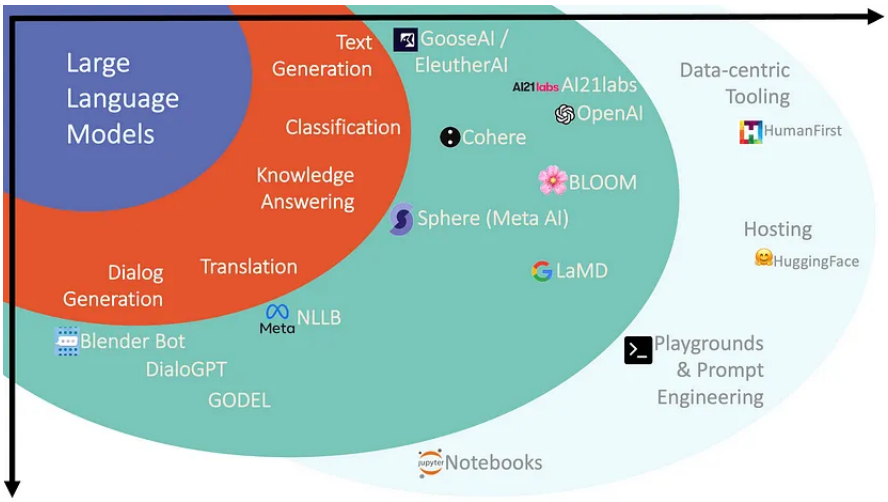

Figure 1: Panoramica del mondo dei Large Language Model.

I contenuti della tesi sono organizzati in cinque capitoli: il primo introduce il natural language processing ed i knowledge graph, il secondo illustra i language model principali, il terzo capitolo illustra le tecnologie impiegate, il quarto il metodo sviluppato ed i risultati sperimentali.

Index

Elenco delle figure

Capitolo 1

Introduzione al Natural Language Processing

1.1 Rappresentazione del Linguaggio Naturale

L'elaborazione del linguaggio naturale, denominata NLP, si riferisce al ramo dell'informatica e, più specificamente, al ramo dell'intelligenza artificiale o AI, che si occupa di dare ai computer la capacità di comprendere testo e parole pronunciate più o meno allo stesso modo degli esseri umani. La NLP[4] combina la linguistica computazionale, ovvero la modellazione basata su regole del linguaggio umano, con modelli statistici, di machine learning e deep learning. La combinazione di queste tecnologie consente ai computer di elaborare il linguaggio umano sotto forma di testo o dati vocali e di "comprenderne" il significato completo, completo dell'intenzione e del sentimento di chi parla o scrive. La capacità degli esseri umani di comprendere le sfumature di un linguaggio è ineguagliabile. Il cervello è in grado di comprendere a fondo la semantica di una parola associando ad essa sensazioni ed esperienze. Un calcolatore non ha questa capacità e nonostante la NLP sia uno strumento potente riscontra ancora una serie di limitazioni e problemi di elaborazione del linguaggio naturale:

- **Comprensione contestuale e omonimi**. Le stesse parole e frasi possono avere significati diversi a seconda del contesto di una frase e molte parole hanno la stessa identica pronuncia ma significati totalmente diversi.
- **Sinonimi**.Parole diverse che esprimono la stessa idea .
- **Ironia e sarcasmo**.Utilizzo di parole e frasi che possono essere positive o negative ma che in realtà connotano il contrario .

- **Ambiguità**.L'ambiguità in NLP si riferisce a frasi che potenzialmente hanno due o più possibili interpretazioni.
  - **Ambiguità lessicale**.Una parola che può essere utilizzata come verbo, sostantivo o aggettivo.
  - **Ambiguità semantica**.l'interpretazione di una frase nel contesto.
- **Errori ortografici**.Sentenze scritte possono contenere errori ortografici, parole errate o usate in modo appropriato possono difatti creare problemi per l'analisi del testo .
- **Colloquialismi e slang**.Frasi informali, espressioni, idiomi e gerghi specifici della cultura rappresentano una serie di problemi per la NLP poichè potrebbero non avere alcuna definizione da dizionario .
- **Linguaggio specifico del dominio**.Diverse aziende ed industrie utilizzano spesso un linguaggio molto diverso fra di loro. Un modello di elaborazione NLP necessario per assistenza sanitaria sarebbe differente da quello utilizzato per elaborare documenti legali.

Indipendentemente dal fatto che la lingua sia parlata o scritta, l'elaborazione del linguaggio naturale utilizza l'intelligenza artificiale per prendere l'input del mondo reale, elaborarlo e dargli un senso in un modo che un computer può comprendere. Ci sono due fasi principali per l'elaborazione del linguaggio naturale: la preelaborazione dei dati e lo sviluppo dell'algoritmo. La preelaborazione dei dati, anche detta preprocessing, implica la preparazione e la pulizia dei dati di testo affinché il computer possa analizzarli. Esistono diversi passaggi per effettuare la pulizia dei dati che variano da attività ad attività, ma i più comuni sono:

- **Segmentazione (tokenization)**. Suddivisione del testo in token.
- **Case Folding**.Conversione di tutte le parole completamente in minuscolo .
- **Normalizzazione**.Questo passaggio ha come obbiettivo quello di ridurre le dimensioni del dizionario utilizzato. Le tecniche di normalizzazione più usate sono:
  - **Stemming**.Ricava da ogni parola la sua radice morfologica.
  - **Lemmatizzazione**.Consiste nel sostituire ciascuna parola con il suo lemma, ovvero la sua forma base presente nel dizionario .

Dopo queste fasi di pulizia dei dati questi appaiono divisi in token ma rimane il problema di insegnare a un computer il significato di questi token. L'idea è quella di creare relazioni tra le varie parole guardando quelle che precedono la parola presa in considerazione e quelle che le seguono. Per rendere possibile ad un computer di comprendere perfettamente il significato di una parola è necessario definirne il contesto C_{w_k} come l'insieme delle N parole prima di w_k e delle M parole successive.

$$C_{w_k} = [w_{k-n}, \ldots, w_{k-1}, w_{k+1}, \ldots, w_{k+m}]$$

Cosi facendo per ogni parola è possibile creare un contesto e in base a questo crare un elenco di caratteristiche utilizzando la frequenza della presenza di altre parole nel suo contesto. Questo porta il computer a creare relazioni tra le parole e queste relazioni sono le chiavi per comprenderne il significato. L'elaborazione del linguaggio naturale risulta utile per qualsiasi applicazione che utilizza del testo. L'elaborazione del linguaggio naturale è utile per le aziende che utilizzano grandi quantità di dati non strutturati in linguaggio umano e hanno bisogno di un modo per elaborarli in modo efficiente.

1.1.1 Bag of Words

Bag of Words è un modello di rappresentazione testuale utilizzato negli algoritmi di machine learning per analizzare testo e documenti. Questo modello mira, come è spiegato nell'articolo [5], ad estrarre caratteristiche dal testo che possono risultare ulteriormente utili per la modellazione. Bag of Words, denominato BOW, aiuta a convertire il testo in rappresentazione numerica, come mostrato nella figura 1.1. La rappresentazione così ottenuta può essere utilizzata per addestrare modelli utilizzando algoritmi di apprendimento. BOW può essere facilmente implementato utilizzando un dizionario Python con ogni chiave impostata su una parola ed ogni valore impostato sul numero di volte che la parola appare in un testo.

Document	the	cat	sat	in	hat	with
the cat sat	1	1	1	0	0	0
the cat sat in the hat	2	1	1	1	1	0
the cat with the hat	2	1	0	0	1	1

Figure 1.1: Esempio di utilizzo della rappresentazione Bag Of Words su tre differenti frasi.

1.1.2 Word Embeddings

Word Embedding è una tecnica di modellazione del linguaggio, introdotta nell'articolo [6], utilizzata per rappresentare le parole o le frasi come vettori di numeri reali in uno spazio di dimensioni inferiori. Le parole vengono raggruppate assieme per ottenere una rappresentazione simile per parole con significato simile. L'idea è quella che parole simili tendono a presentarsi insieme ed avranno quindi un contesto simile, ad esempio: "la mela è un frutto" e "il mango è un frutto", la mela ed il mango tendono ad avere un contesto simile, ad esempio la frutta. Viene utilizzata una matrice di co-occorrenza che contiene parole sia nelle righe che nelle colonne. Una visualizzazione grafica della mtrice di co-occorrenza può essere trovata alla figura 1.2.

	NLP	**flying**	**I**	**like**	**deep**	**learning**	**enjoy**
NLP	0.0	0.0	0.0	1.0	0.0	0.0	0.0
flying	0.0	0.0	0.0	0.0	0.0	0.0	1.0
I	0.0	0.0	0.0	2.0	0.0	0.0	1.0
like	1.0	0.0	2.0	0.0	1.0	0.0	0.0
deep	0.0	0.0	0.0	1.0	0.0	1.0	0.0
learning	0.0	0.0	0.0	0.0	1.0	0.0	0.0
enjoy	0.0	1.0	1.0	0.0	0.0	0.0	0.0

Figure 1.2: Matrice di co-occorrenza relativa a tre frasi, ovvero:I enjoy flying, I like NLP e I like deep learning.

Lo scopo di questa matrice è quello di indicare il numero di volte in cui la parola nella riga i appare nel contesto della parola della colonna j. In questo modo è possibile creare un vettore ad alta dimensione che porta per ogni parola informazioni sulle relazioni con altre parole. Di conseguenza, per utilizzare una matrice di co-occorrenza, devono essere definite sia le entità che il contesto in cui esse si verificano, ovvero la frase in cui esse appaiono. Al fine di aumentare l'efficienza vengono utilizzate tecniche avanzate come la Singular vector decomposition, anche chiamata SVD. La SVD, viene utilizzata per ridurre la dimensionalità dei vettori associati a ciascuna parola cercando di perdere la minore quantità possibile di informazione. Tramite SVD viene applicato il processo di Latent Semantic Model tramite il quale vengono sottolineate le relazioni tra le varie parole, ad esempio tra capitali e nazioni (il risultato di questo esempio in particolare è visualizzabile alla figura 1.3.

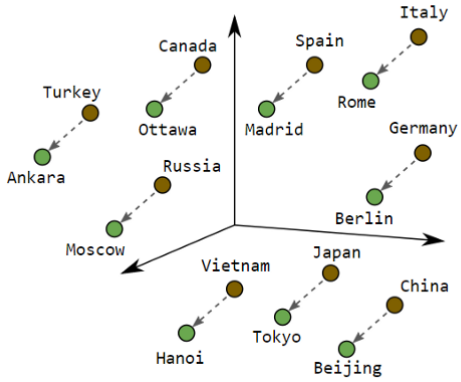

Figure 1.3: Rappresentazione vettoriale per rappresentare le relazioni tra nazioni e capitali.

Questa fase viene anche chiamata riduzione della dimensionalità e mira a creare uno spazio vettoriale a dimensione ridotte che contenga parole simili messe insieme e parole diverse poste ad una certa distanza. La tecnica di word embeddings in NLP ha migliorato la capacità dei computer di comprendere i contenuti basati su un testo in un modo migliore. È considerata una delle scoperte più significative del deep learning per la risoluzione di problemi complessi per l'elaborazione del linguaggio naturale. Approcci comuni per il calcolo dei vettori di parole includono:

- **Word2Vec**.Il primo approccio che utilizzava efficientemente le reti neurali per apprendere gli incorporamenti da un grande set di dati .
- **fastText**.Un efficiente modello basato sui caratteri in grado di elaborare parole fuori dal vocabolario .
- **ELMo**.Rappresentazioni di parole contestualizzate in grado di gestire la polisemia(parole con significati multipli in contesti diversi).

1.1.3 Sentence Embeddings

Sentence embeddings è il nome di un insieme di tecniche dell'elaborazione del linguaggio naturale in cui le frasi sono mappate su vettori di numeri reali. Questi vettori vengono utilizzati per catturare una gamma di relazioni semantiche tra frasi come somiglianze, contraddizione e implicazione. Le tecniche di sentence embedding possono essere voste come un evoluzione di quelle di word embedding. Nel caso di testo di grandi dimensioni utilizzare solo parole sarebbe molto laborioso e le informazioni che sarebbe possibile estrarre

sarebbero limitate. Ad esempio assumendo di imbattersi in una frase come "non mi piacciono i posti affollati" seguita da "tuttavia mi piace una delle città più trafficate del mondo, New York" usare la tecnica di word embeddings risulterebbe insufficiente. Difatti per estrarre l'inferenza tra "luoghi affollati" e "città trafficate" la tecnica migliore risulta essere quella di sentence embeddings. La rappresentazione di intere frasi come vettori, inoltre, aiuta il calcolatore a capire il contesto, l'intenzione e altre sfumature all'interno del testo, come si può vedere alla figura 1.4. Sono stati proposti numerosi approcci per sviluppare la tematica del sentence embeddings e affrontare problemi come la polisemia, le parole fuori dal vocabolario o la conservazione dell'ordine delle parole. I metodi più utilizzati, ad oggi, sono:

- **Doc2Vec**.Algoritmo che si basa su Word2vec. Segue il presupposto che il significato di una parola sia dato dalle parole che le appaiono vicine .
- **SentenceBERT**.Considerato l'algoritmo all'avanguardia per ceare mappature sulle frasi .

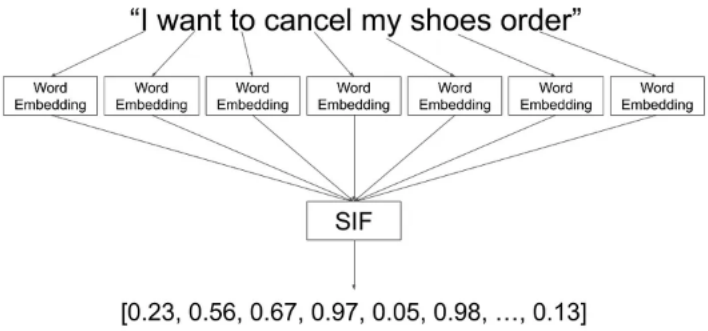

Figure 1.4: Un esempio di sentence embeddings che utilizza una media ponderata degli incorporamenti di ogni parola e applica una riduzione di dimensionalità per ottenere l'incorporamento della frase.

1.1.4 Named Entity Recognition

Il riconoscimento di entità[7], anche chiamato NER, è una forma di elaborazione del linguaggio naturale che prevede l'estrazione e l'identificazione di informazioni essenziali dal testo. Le informazioni che vengono estratte e classificate sono denominate entità. Un'entità può essere una singola parola o una serie di parole che si riferisce costantemente alla stessa cosa. Utilizzando

NER è possibile comprendere l'argomento o il tema di un testo di grandi dimensioni e di raggruppare rapidamente i testi in base alla loro pertinenza e somiglianza. Un semplice esempio del funzionamento di un algoritmo di NER è visualizabile alla figura 1.5.

Figure 1.5: Esempio di funzionamento di un semplice algoritmo di NER.

Gli scenari dove gli algoritmi di NER vengono utilizzati, sono:

- **Servizio clienti**.NER aiuta a ridurre i tempi di risposta classificando le richieste degli utenti, i reclami e le domande filtrate per parole chiave specifiche.
- **Assistenza sanitaria**.Aiuta i medici a comprendere rapidamente i referti estraendo le informazioni essenziali, riducendo quindi i carichi di lavoro e migliorando gli standard di cura dei pazienti.
- **Motori di ricerca**.Migliora la velocità la pertinenza dei risultati della ricerca analizzando le query di ricerca e altri testi.
- **Risorse umane**.Migliora i flussi di lavoro interni classificando i reclami e le domande dei dipendenti e accelera il processo di assunzione riassumendo i CV dei candidati.

Gli algoritmi NER tradizionali includevano solo nomi, luoghi ed organizzazioni. Tuttavia, ora possono essere addestrati dinamicamente per estrarre più delle sole entità menzionate in precedenza. NER è un approccio semplice ma efficace per ridurre la ricerca di uno spazio di stato indirizzando l'algoritmo a pesare maggiormente le frasi se viene trovata una porzione di entità. Qualsiasi algoritmo NER consiste in due fasi:

- **Rilevazione di entità**. Consiste nel rilevare una parola o una stringa di parole che formano un'entità. Ogni parola rappresenta un token, e un

singolo token o un insieme di token identifica un'entità. Il tagging dentro-fuori-inizio è un modo comune per indicare dove iniziano e finiscono le entità.

- **Classificazione delle entità rilevate**. Richiede la creazione di categorie nelle quali inserire le entità rilevate in precedenza.

1.2 Deep Neural Network in NLP

Una Deep Neural Network, anche chiamata DNN, è una rete neurale con un alto livello di complessità, [8]. Una visualizzazione della struttura è disponibile alla figura 1.6. Possiamo immaginare queste reti schematizzate in tre strati ciascuno costituito da migliaia di neuroni e decine di migliaia di connessioni:

- **input**. Strato di ingresso nel quale arrivano i segnali che vengono elaborati in modo da adattarsi alle richieste dei neuroni della rete.
- **hidden**. Questo strato deve essere costituito da due o più livelli ed è qui che avviene il processo di elaborazione dei dati ricevuti . Il concetto di profondo delle DNN si riferisce esattamente a questo strato e al suo "spessore".
- **output**. Nello strato di uscita i risultati dell'elaborazione avvenuta nello strato hidden vengono raccolti e adattati alle richieste del successivo blocco di rete neurale.

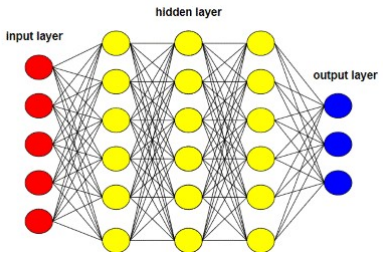

Figure 1.6: Composizione di una Deep Neural Network.

Le reti neurali profonde vengono spesso impiegate per gestire dati non etichettati e non strutturati offrendo prestazioni all'avanguardia nel campo dell'apprendimento automatico e dell'intelligenza artificiale. È alle Deep Neural Network che si deve il rapido progresso dei sistemi di analisi automatica di immagini, video, audio e serie temporali, alla base del deep learning. Le reti neurali profonde hanno raggiunto performance all'avanguardia e continuano giorno

dopo giorno a crescere e migliorare i risultati ottenuti. L'idea generale di una DNN è che apprende attraverso una serie di esempi, così facendo, la rete impara allo stesso modo del cervello umano, attraverso la pratica e commettendo errori. Le DNN sono diventati uno strumento potente e prezioso in molte applicazioni industriali e commerciali, come ad esempio le macchine automatiche di traduzione, la classificazione di oggetti e la generazione automatica di didascalie. Nei paragrafi successivi verrà spiegata l'applicazione della DNN al natural language processing.

1.2.1 Word Embeddings con DNN

La sfida più grande nel campo dell'elaborazione del linguaggio naturale è trovare un modo per rappresentare il significato di una parola. L'incorporamento delle parole consiste nella rappresentazione delle parole in uno spazio vettoriale che ne mostri le relazioni e che venga utilizzato per catturare le informazioni sintattiche e semantiche delle parole. Quando una parola appare in una frase il suo contesto è l'insieme o la sequenza di parole che appaiono nelle vicinanze. La creazione di Word Embedding è un argomento molto studiato e discusso nel mondo informatico. Un ottimo modo per generare questi vettori è sfruttare le Deep Neural Networks, utilizzando il contesto di una parola, chiamata $Context_{w_0}$. Le reti neurali profonde sono addestrate a ricercare la parola corretta appartenente ad uno specifico contesto, ad esempio mascherando una parola in una frase. In questo caso il modello deve generare una distribuzione di probabilità sul dizionario per indovinare la parola mascherata leggendo le parole prima e quelle dopo di quella mascherata. Utilizzando un set di dati di grandi dimensioni e molto tempo di addestramento, il modello può ottenere ottimi risultati. Tramite questo approccio la rete neurale apprende internamente in che modo modellare il linguaggio. Le reti profonde sono definite tali per la presenza degli strati nascosti, questi strati contengono ciò che il modello ha appreso sul linguaggio e ci permettono di creare vettori da utilizzare per l'incorporamento di parole.

1.2.2 Importanza della contestualizzazione per Word Embeddings

Riferendosi all'esempio fatto in precedenza il modello può leggere le parole precedenti e successive a quella mascherata generando una distribuzione di probabilità sul dizionario. Definiamo $X_{0:T} = (x_0, x_1, \ldots, x_T)$ il dizionario della lingua, w_j la parola mascherata e $C_j = (w_0, \ldots, w_{j-1}) \cup (w_{j+1}, \ldots, w_n)$ il suo contesto. Quindi il modello cerca di prevedere la giusta distribuzione di

probabilità su $X_{0:T}$, nella modalità seguente:

$$P(X_{0:T}|C_j)$$

. Dalla definizione appena data possiamo vedere che il contesto ha un'importante influenza sulla previsione della parola mascherata. Difatti è risaputo che una stessa parola usata in frasi diverse genera diversi incorporamenti di parole. Tramite le deep neural netwok si ottiene una rappresentazione vettoriale fortemente legata al contesto, detta anche Contextual Word Embedding. Questa rappresentazione fornisce miglioramenti significativi sui risultati ottenibili in varie attività come ad esempio classificazione o raccomandazioni.

1.3 Knowledge Graph

Un grafo della conoscenza viene creato quando applichi un'ontologia (il nostro modello di dati) a un insieme di dati (ad esempio dati di libro, autore ed editore). Prima di poter definire un knowledge graph è necessario introdurre il concetto di ontologia.

1.3.1 Ontologia

Le ontologie sono modelli di dati generalizzati, nel senso che modellano solo tipi generali di entità che condividono determinate proprietà, ma non includono informazioni su individui specifici nel nostro dominio. Ad esempio, invece di descrivere il tuo cane, Spot, e tutte le sue caratteristiche individuali, un'ontologia dovrebbe concentrarsi sul concetto generale di cani, cercando di catturare le caratteristiche che la maggior parte/molti cani potrebbero avere. Ciò ci consente di riutilizzare l'ontologia per descrivere altri cani in futuro. Ci sono tre componenti principali di un'ontologia, che di solito sono descritti come segue:

- **Classi**. I tipi distinti di cose che esistono nei nostri dati.
- **Relazioni**. Proprietà che collegano due classi .
- **Attributi**. Proprietà che descrivono una singola classe .

Le ontologie rappresentano la spina dorsale della semantica formale di un grafo della conoscenza. Fungono da contratto formale tra gli sviluppatori del grafo della conoscenza e i suoi utenti per quanto riguarda il significato dei dati in esso contenuti, come nell'esempio 1.7.

Il Web Ontology Language (OWL)[9] è un esempio di ontologia ampiamente adottata, supportata dal World Wide Web Consortium (W3C), una comunità

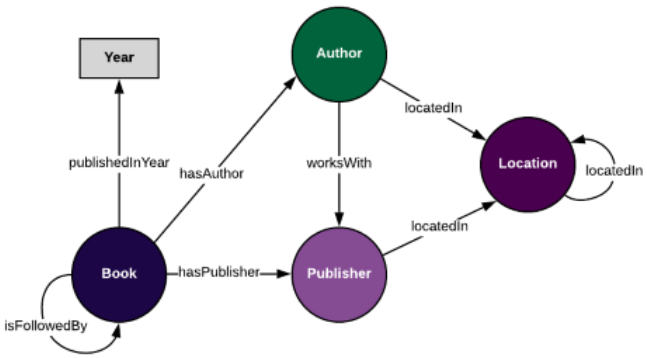

Figure 1.7: Esempio di ontologia su libri ed i relativi autori.

internazionale che sostiene gli standard aperti per la longevità di Internet. In definitiva, questa organizzazione della conoscenza è supportata da infrastrutture tecnologiche come database, API e algoritmi di apprendimento automatico, che esistono per aiutare le persone e i servizi ad accedere ed elaborare le informazioni in modo più efficiente.

1.3.2 Definizione di Knowledge graph

La nozione di Knowledge Graph nasce da progressi scientifici in diverse aree di ricerca come il web, i database, la rappresentazione della conoscenza e NLP. Un grafo della conoscenza viene creato quando si applica un'ontologia (il modello dei dati) ad un insieme di dati individuali (come ad esempio i dati di libro, autore ed editore). Un grafo della conoscenza, che appare graficamente

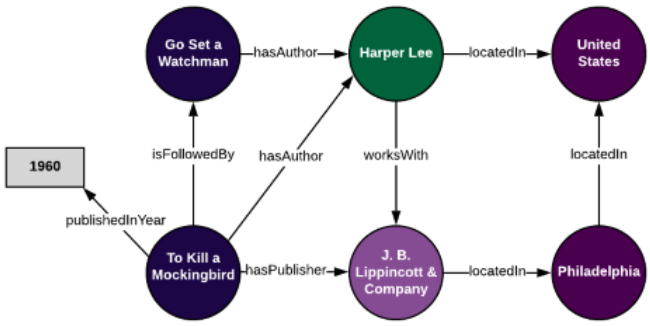

Figure 1.8: Creazione di un knowledge graph sull'esempio di ontologia precedente.

come nella figura 1.8, noto anche come rete semantica, rappresenta una rete di

entità del mondo reale, ad es. oggetti, eventi, situazioni o concetti e illustra le relazioni tra di essi. Queste informazioni sono solitamente memorizzate in un database a grafo e visualizzate come una struttura a grafo, da cui il termine "grafo" della conoscenza". Un grafo della conoscenza è costituito da tre componenti principali: nodi, bordi ed etichette. Qualsiasi oggetto, luogo o persona e quindi in generale qualsiasi entità può essere un nodo. Un bordo collega una coppia di nodi e cattura la relazione di interesse tra loro, ad esempio, relazione di amicizia tra due persone, relazione cliente tra un'azienda e una persona o una connessione di rete tra due computer. Le etichette catturano il significato della relazione, ad esempio la relazione di amicizia tra due persone. Più formalmente, dato un insieme di nodi N e un insieme di etichette L, un grafo della conoscenza è un sottoinsieme del prodotto incrociato N $\times$ L $\times$ N. Ciascun membro di questo insieme è indicato come un triplo e può essere visualizzato come mostrato di seguito. I grafi della conoscenza sono in genere costituiti da set di dati provenienti da varie fonti, che spesso differiscono nella struttura. Schemi, identità e contesto lavorano insieme per fornire una struttura a dati diversi. Questi componenti aiutano a distinguere le parole con significati multipli. Ciò consente ai prodotti, come l'algoritmo del motore di ricerca di Google, di determinare la differenza tra Apple, il marchio, e apple, il frutto. I grandi grafici forniscono conoscenze di base, concetti simili a quelli umani e consapevolezza dell'entità, per consentire un'interpretazione più accurata del testo; I linguaggi di query su grafi supportano non solo operatori relazionali standard (join, unioni, proiezioni, ecc.), ma anche operatori di navigazione, operatori per trovare entità connesse tramite percorsi di lunghezza arbitraria.

1.3.3 Applicazioni dei Knowledge graph

Esistono numerose applicazioni[10] dei grafi della conoscenza sia nel campo della ricerca che a livello industriale. A livello informatico, l'utilizzo di una rappresentazione grafica diretta è ampiamente diffuso, viene ad esempio impiegato per grafici del flusso di dati, diagrammi di decisione binaria, grafici di stato, ecc

Tuttavia, i grafi della conoscenza hanno anche applicazioni in altri settori, come: -Vendita al dettaglio: i grafi della conoscenza sono stati utilizzati per strategie di up-sell e cross-sell, raccomandando prodotti in base al comportamento di acquisto individuale e alle tendenze di acquisto popolari tra i gruppi demografici. -Intrattenimento: i grafi della conoscenza vengono sfruttati anche per i motori di raccomandazione basati sull'intelligenza artificiale (AI) per piattaforme di contenuti, come Netflix, SEO o social media. Sulla base dei clic e di altri comportamenti di coinvolgimento online, questi provider consigliano nuovi contenuti da leggere o guardare dagli utenti.

-Finanza: questa tecnologia è stata utilizzata anche per iniziative di know-your-customer (KYC) e antiriciclaggio nel settore finanziario. Aiutano nella prevenzione e nelle indagini sui crimini finanziari, consentendo agli istituti bancari di comprendere il flusso di denaro attraverso la loro clientela e identificare i clienti non conformi.

-Sanità: i grafici della conoscenza stanno anche avvantaggiando il settore sanitario organizzando e classificando le relazioni all'interno della ricerca medica. Queste informazioni aiutano i fornitori convalidando le diagnosi e identificando i piani di trattamento in base alle esigenze individuali.

1.4 Obbiettivi di questo progetto

L'obiettivo di questo progetto è di proporre un nuovo metodo per integrare le informazioni sui termini biomedici contenute in grafo di conoscenza esterno, con del testo scritto. Vogliamo poi studiare come i modelli di information retrieval si comportano utilizzando questa integrazione per espandere l'informazione semantica contenuta nelle query. E' noto a livello di letteratura che una maggiore informazione espressa nella query porta a una più precisa selezione dei documenti. Pertanto, vogliamo verificare in quali casi e circostanze espandere una query con informazioni da un KG porta a risultati migliori. In particolare, il nuovo metodo proposto consiste nel selezionare le parole biomediche più importanti, tramite l'utilizzo di NER neurali biomedici, ed espandere questi termini utilizzando un knowledge graph di dominio. Dopo di che il sottografo selezionato verrà inserito nella query tramite un processo di linearizzazione prompt-based. Saranno effettuati degli studi nell'ambito del deep learning applicato al Natural Language Processing, concentrandoci sui large language model che oggi rappresentano lo stato dell'arte tra le tecnologie per l'elaborazione del testo[11]. . Faremo esperimenti per trovare gli hyperparametri ottimali per questi modelli, dopo di che procederemo ad applicare questi modelli all'input arricchito. In questa maniera vogliamo verificare se il nuovo input contribuisce ad un miglioramento delle performance. Tutti gli esperimenti saranno eseguiti su un dataset di document retrieval BioASQ task8, basato sui documenti pubmed.

Capitolo 2

Related Works

2.1 Attention

Ponendo l'attenzione sul significato della parola inglese "Attention", si può notare che significa dirigere l'attenzione su qualcosa di specifico e concentrarsi su esso. Il meccanismo di attenzione, spiegato nell'articolo [12], è stato introdotto per migliorare le prestazioni del modello codificatore-decodificatore per la traduzione automatica. L'idea alla base del meccanismo di attenzione era quella di consentire al decodificatore di utilizzare le parti più rilevanti della sequenza di input in modo flessibile, mediante una combinazione ponderata di tutti i vettori di input codificati, con i vettori più rilevanti a cui venivano attribuiti i pesi più alti. Il meccanismo di attenzione è stato introdotto per affrontare il problema del collo di bottiglia che si presenta con l'uso di un vettore di codifica a lunghezza fissa, in cui il decodificatore avrebbe un accesso limitato alle informazioni fornite dall'input. Per sequenze lunghe ciò diventa particolarmente problematico dato che la dimensionalità per la loro rappresentazione sarebbe costretta ad essere la stessa di sequenze più brevi e semplici. Prima di procedere è necessario specificare le quattro componenti fondamentali in qualsiasi meccanismo di attenzione:

- **Query**. La query è un vettore di caratteristiche che descrive ciò che si ricerca all'interno della sequenza.
- **Chiavi**. Per ogni elemento di input vi è una chiave che è un vettore di caratteristiche. Questo vettore di funzionalità descrive approssimativamente ciò che l'elemento di input offre.
- **Valori**. Per ogni elemento di input, oltre un vettore di caratteristiche c'è anche un vettore di valore. Questo vettore di caratteristiche è quello su cui si vuole calcolare la media.

- **Funzione di punteggio**. Funzione utilizzata per valutare a quali elementi è necessario prestare attenzione. La funzione di punteggio prede in input la query e una chiave e restituisce il punteggio, ovvero l'attenzione, della coppia query-chiave.

2.1.1 Prima implementazione

Il primo meccanismo di attenzione proposto calcola, passo dopo passo:

- **Punteggi di allineamento**. Il modello di allineamento prende gli strati nascosti codificati, h_i, e l'output del decodificatore precedente, s_{t-1} , per calcolare un punteggio, $e_{t,i}$, che indica quanto bene gli elementi della sequenza di input si allineino con l'output corrente in posizione t . Il modello di allineamento è rappresentato da una funzione, a(.), che può essere implementata da una rete neurale feedforward. L'assegnazione dei punteggi di allineamento avviene come nell'immagine 2.1.

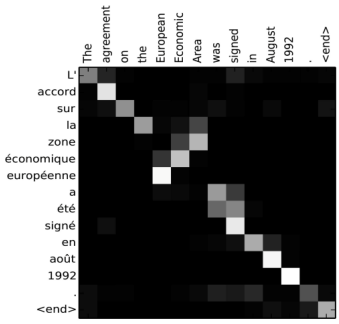

Figure 2.1: Esempio di assegnazione del punteggio di allineamento che quantifica la quantità di attenzione che il decodificatore deve porre su ciascun elemento.

- **Pesi**. I pesi, α_{ti}, vengono calcolati applicando un'operazione softmax ai punteggi di allineamento calcolati in precedenza.

- **Vettore di contesto**. Un unico vettore di contesto, c_t, viene immesso nel decodificatore ad ogni passo temporale. È calcolato da una somma ponderata di tutti i T strati nascosti del codificatore.

$$c = \sum_i \alpha_{ti} h_i \tag{2.1}$$

Tuttavia il meccanismo di attenzione può essere riformulato in una forma generale che può essere applicata a qualsiasi attività del tipo sequence-to-sequence in cui le informazioni potrebbero non essere necessariamente correlate in modo sequenziale.

2.1.2 Generalizzazione meccanismo di Attention

Il meccanismo di attenzione generalizzato si avvale di tre componenti principali, ovvero le query (Q), le chiavi (K), e i valori (V), e si svolge secondo i seguenti passaggi:

- ogni vettore di query, q=s_{t-1}, viene confrontato con un database di chiavi per calcolare un valore di punteggio, che viene calcolato come il prodotto scalare della query specifica in esame, q, con ciascun vettore chiave, k_i:

$$e_q, k_i = q * k_i \tag{2.2}$$

- I punteggi vengono elaborati attraverso una funzione softmax per generare i pesi:

$$\alpha_q, k_i = softmax(e_q, k_i) \tag{2.3}$$

- L'attenzione generalizzata viene quindi calcolata da una somma ponderata dei vettori di valori, v_{k_i}, dove ogni vettore di valore è accoppiato con la corrispondente chiave:

$$attention(q, K, V) = \sum_i \alpha_{q,k_i} v_{k_i} \tag{2.4}$$

Il meccanismo di attenzione è emerso come un miglioramento rispetto al sistema di traduzione automatica basato sull'architettura encoder-decoder nell'elaborazione del linguaggio naturale. Successivamente, questo meccanismo, o una delle sue varianti, è stato utilizzato in altre applicazioni, tra cui visione artificiale, elaborazione vocale, ecc. L'attenzione collega il codificatore e il decodificatore e fornisce al decodificatore informazioni da ogni strato nascosto del codificatore. Con questo framework, il modello è in grado di concentrarsi selettivamente su parti preziose della sequenza di input e quindi apprendere l'associazione tra di esse. Questo aiuta il modello a far fronte in modo efficiente a frasi di input lunghe.

2.1.3 Multi-Head Attention

Estendendo i meccanismi di attenzione a più teste, come spiegato in [13], ovvero triplette query chiave e valore, sulle stesse funzionalità si migliorano le prestazioni del modello espandendone le capacità. Il meccanismo di attenzione multi-head, la cui struttura può essere visualizzata graficamente alla figura 2.2, consente di utilizzare congiuntamente diversi sottospazi di rappresentazione di query, chiavi e valori, trasformando questi ultimi con h proiezioni lineari apprese indipendentemente. Alla fine, gli output di ciascuno degli h livelli di

attenzione vengono concatenati e trasformati con un'altra proiezione lineare al fine di produrre l'output finale con la corretta dimensione. Sostanzialmente ad ogni livello di attenzione h viene calcolato l'output Z_i utilizzando una tripletta W_k, W_v e W_q. Infine tutti gli output Z vengono concatenati e moltiplicati per una matrice W^o che è stata addestrata insieme al modello in modo tale da creare un vettore delle giuste dimensioni:

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h) * W_o \tag{2.5}$$

$$dovehead_i = Attention(Q(W_i)^q, K(W_i)^k, V(W_i)^v) \tag{2.6}$$

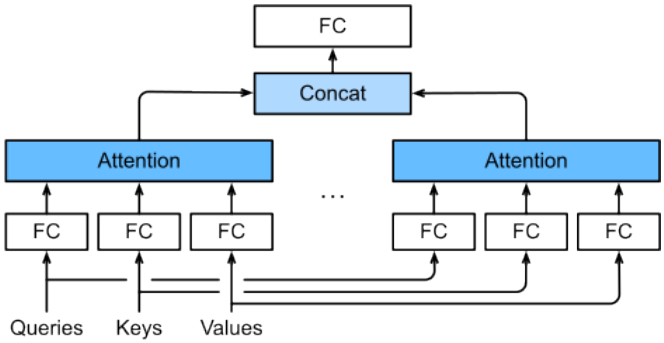

Figure 2.2: Multi Head Attention dove più teste sono concatenate e poi linearmente trasformate.

2.1.4 Self Attention

La self attention, anche detta auto-attenzione, si riferisce alla capacità di un modello di prestare attenzione a diverse parti della sequenza di input quando si effettuano previsioni. Il nome deriva dal fatto l'auto attenzione consente ad un modello di occuparsi di diverse parti della stessa sequenza di input, mentre l'attenzione "regolare" consente di occuparsi di parti diverse di un'altra sequenza. Il meccanismo di self-attention[14] permette di guardare l'intero contesto della sequenza considerata mentre si codifica ciascuno degli elementi di input e proprio per questo ha apportato notevoli miglioramenti nell'affrontare operazioni su sequenze molto lunghe. Le sequenze di input vengono elaborate una ad una come segue:

- **1**. Viene effettuata la moltiplicazione di ciascuno dei vettori di input del codificatore con tre matrici di pesi (W_q, W_k, W_v) producendo tre vettori: il vettore chiave, il vettore query e il vettore valore.

- **2**. Il vettore di query dell'input corrente viene moltiplicato con i vettori di chiave da altri input.
- **3**. Il risultato precedentemente ottenuto viene diviso per la radice quadrata delle dimensioni del vettore chiave.
- **4**. Si svolge l'applicazione della funzione softmax a tutti i punteggi di auto-attenzione calcolati rispetto alla parola oggetto di studio.
- **5**. Il vettore valore viene moltiplicati per il vettore calcolato al passo precedente.
- **6**. I valori ponderati ottenuti al quinto passggio vengono sommati ottenendo l'output della self attention per la parola presa in input.

Matematicamente parlando per delle matrici di input Q, K, V, l'output della self-attention viene calcolato così:

$$a = Softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.7}$$

2.2 Transformers

Descritti per la prima volta in un documento del 2017 di Google, i Transformer sono tra le classi di modelli più recenti e potenti inventate fino ad oggi. Un Transformer è un tipo di architettura di rete neurale che evita la ricorrenza e si affida interamente ad un meccanismo di attenzione per disegnare dipendenze globali tra input e output. Si supponga, ad esempio, di voler costruire un modello che sia in grado di tradurre un testo da una lingua ad un'altra. Le soluzioni generalmente adottate per effettuare la traduzione portano con sé diverse difficoltà:

- *Lunghezza variabile del testo.*
- *Enorme dimensione dei dati dopo l'applicazione del modello Bag-of-Words.*
- *Costo di calcolo elevato.*

Per far fronte ai precedenti problemi è risultato necessario adottare un nuovo modello, che non fosse dipendente dall'utilizzo della rappresentazione Bag-of-Words. È qui che entra in gioco il Transformer, ovvero il primo modello di traduzione che fa affidamento interamente sulla self-attention per calcolare le rappresentazioni del suo input e output, risultando più parallelizzabile, veloce e richiedendo un minor tempo per la fase di allenamento. Invece di elaborare

i token separatamente, il testo viene suddiviso in segmenti in modo tale da apprendere le dipendenze tra di loro. Questo modello è stato progettato sulla base di un'architettura encoder-decoder. In un Transformer, i token di input vengono convertiti in vettori, ai quali vengono aggiunte alcune informazioni sulla posizione per tenere conto dell'ordine dei token durante l'elaborazione simultanea del modello. I Transformers[15] vengono utilizzati non solo per l'attività di traduzione ma per qualsiasi attività che trasforma una sequenza di input in una sequenza di output, come ad esempio il riconoscimento vocale e la trasformazione da testo a voce.

2.2.1 Architettura del modello

L'architettura Transformer, visualizzabile alla figura 2.3, si basa su una struttura codificatore-decodificatore ma non utilizza ricorrenza e convoluzioni per generare un output. In poche parole il compito del codificatore è quello di mappare una sequenza di input di rappresentazioni di simboli(x_0, ...,x_n) ad una sequenza di rappresentazioni continue di output(z_0, ...,z_n). Questi vettori rappresentano l'input del decodificatore che generare una sequenza di uscita y_0, ...,y_n. Il modello risulta essere autoregressivo, consumando i simboli generati in precedenza come input aggiuntivo durante la generazione del successivo. Il Transformer crea questa architettura utilizzando self-attenction, e strati completamente collegati sia per il codificatore che per il decodificatore.

2.2.2 Encoder

L'encoder, anche chiamato codificatore, utilizzato dal Transformer si compone di sei blocchi di codifica impilati. Gli output dell'ultimo blocco del codificatore diventano l'input per il decodificatore. Le sequenze di input passate al decodificatore possono essere descritte come incorporamenti arricchiti attraverso meccanismi di self-attention multi-head e reti feed-forward basate sulla posizione con connessioni residue e normalizzazione di livello. I sei livelli di codifica del Transformer applicano le stesse trasformazioni lineari a tutte le parole nell sequenza di input, ma ogni livello utilizza pesi diverse(W_1, W_2) e diversi parametri (b_1, b_2) per farlo:

$$FFN(x) = ReLU(W_1x + b_1)W_2 + b_2 \tag{2.8}$$

I blocchi di codifica, inoltre, utilizzano connessioni residuali, (layernorm(.)), seguite da una normalizzazione del livello (sublayer(x)):

$$layernorm(x + sublayer(x)) \tag{2.9}$$

La normalizzazione del livello mira a impedire lo spostamento della media e della deviazione standard dell'incorporamento di elementi vettoriali.

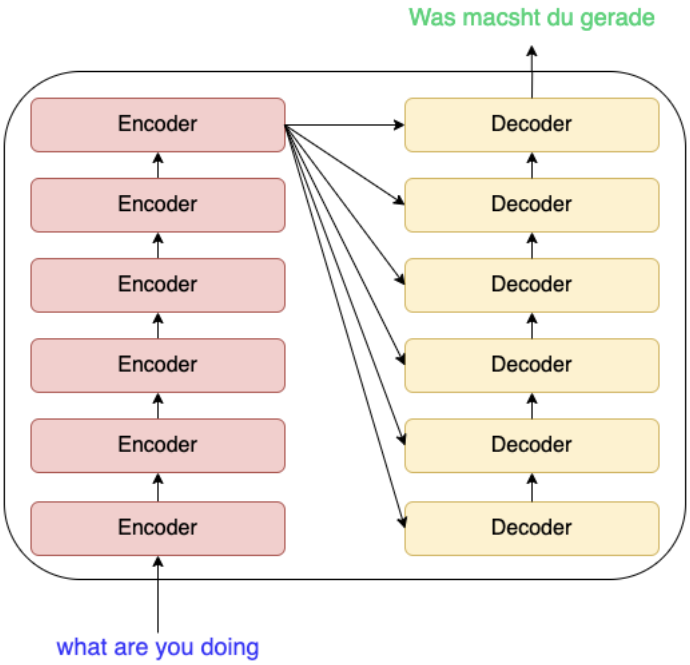

Figure 2.3: Arhitettura di un modello Transformer con 6 blocchi di codifica e decodifica utilizzato per la traduzione da spagnolo a tedesco.

2.2.3 Decoder

Il decoder, anche chiamato decodificatore, condivide diverse caratteristiche con il codificatore. Il decoder del modello Transformer si compone di sei blocchi di decodifica, ciascuno formato da tre sottolivelli:

- Il primo sotto-livello riceve l'output precedente dello stack del decodificatore, lo aumenta con informazioni sulla posizione e implementa su di esso il meccanismo di auto-attenzione multi-head mascherata. La maschera, applicata alle matrici Q e K, viene implementata sopprimendo i valori di matrice che altrimenti corrisponderebbero a connessioni illlegali.

- Il secondo sotto-livello implementa un meccanismo di auto-attenzione multi-head simile a quello implementato nel primo sottolivello del codificatore. Sul lato del decodificatore, questo meccanismo riceve le query dal precedente sottolivello del decodificatore e le chiavi ed i valori dall'output del codificatore.

- Il terzo sotto-livello implementa una rete feed-forward completamente connessa.

Inoltre anche i tre sotto-strati del blocco di codifica hanno connessioni residue e sono seguiti da uno strato di normalizzazione. Le differenze tra encoder e

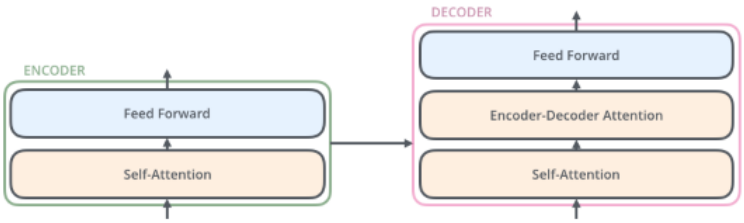

Figure 2.4: Strati di formazione di encoder e decoder.

decoder sono graficamente visibili consultando l'immagine 2.4.

2.2.4 Attention e Transformers

Il primo passaggio consiste nel calcolare le matrici **Query**, **Key** e **Value**. Queste ultime sono ottenute impacchettando gli incorporamenti in una matrice X e moltiplicandola per le matrici di peso che addestrate in precedenza (W_q, W_k, W_v). Infine si utilizza una formula per calcolare gli output Z dello strato di auto-attenzione:

$$Z = SoftMax(\frac{QK^T}{\sqrt{\vec{d_k}}})V \tag{2.10}$$

2.2.5 Codifica posizionale

La posizione e l'ordine delle parole all'interno di una frase sono conoscenze essenziali per poter comprendere un qualsiasi linguaggio. Il problema è che l'architettura Transformer non ha alcuna conoscenza della posizione di ogni parola all'interno di un testo, di conseguenza, vi è la necessità di un modo per incorporare l'ordinamento delle parole all'interno del modello. A tal fine viene utilizzata la codifica posizionale[16], che fornisce una rappresentazione della posizione di ogni parola all'interno di una sequenza. La codifica posizionale aggiunge informazioni sulla posizione di una parola nella frase di input utilizzando funzioni trigonometriche:

$$PE(pos, 2i) = sin(\frac{pos}{1000^{\frac{2i}{dmodel}}}) \tag{2.11}$$

$$PE(pos, 2i+1) = cos(\frac{pos}{1000^{\frac{2i}{dmodel}}}) \tag{2.12}$$

È importante sottolineare che questa codifica non è integrata nel modello stesso, bensì è un'informazione posizionale inserita in ogni incorporamento di parole alla fine degli stack del codificatore e del decodificatore.

2.2.6 Feed Forward Network

Ciascun livello all'interno dell' encoder e del decoder contiene una rete feed-forward completamente connessa. Una rete neurale feed forward è una rete neurale artificiale strutturata in strati interconnessi, ognuno formato da neuroni. Il modello feed forward è la forma più semplice di rete neurale poiché le informazioni vengono elaborate solo in una direzione. Sebbene i dati possano passare attraverso più nodi nascosti, si muovono sempre in una direzione e mai all'indietro. All'interno dell'architettura Transformer lo strato di feed-forward si trova come un sottolivello, sia nel codificatore che nel decodificatore, appena dietro al sottolivello di self-attention. È una trasformazione basata sulla posizione che consiste in una trasformazione lineare, ReLU, combinata con un'altra trasformazione lineare. Matematicamente parlando può essere rappresentata come segue:

$$FFN(x) = max(0, YW_1 + b_1)W_2 + b_2 \tag{2.13}$$

Consultando la figura 2.5 è possibile visualizzare l'architettura completa di un Transformer, compreso di Feed Forward Network.

2.2.7 Sentence Transformer

Sentence Transformers è un framework Python per incorporamento di sentenze, testo, immagini e frasi all'avanguardia. È possibile utilizzare questo framework per calcolare incorporamenti di frasi/testo per più di 100 lingue. Il testo viene incorporato nello spazio vettoriale in modo tale che un testo simile sia vicino e possa essere trovato in modo efficiente utilizzando la somiglianza coseno. Decine di migliaia di utenti fanno affidamento su di esso per creare sistemi per la classificazione del testo, la ricerca neurale/semantica, il clustering del testo e altre attività di AI. Il framework è basato su PyTorch e Transformers e offre un'ampia raccolta di modelli pre-addestrati ottimizzati per varie attività. I modelli sono basati su reti di Transformer come BERT / RoBERTa ecc... e raggiungono prestazioni all'avanguardia in vari compiti. Le pipeline sono API che risultano necessarie per poter utilizzare un Sentence Transformer in diverse attività, tra cui Named Entity Recognition, Masked Language Modeling, Question Answering.

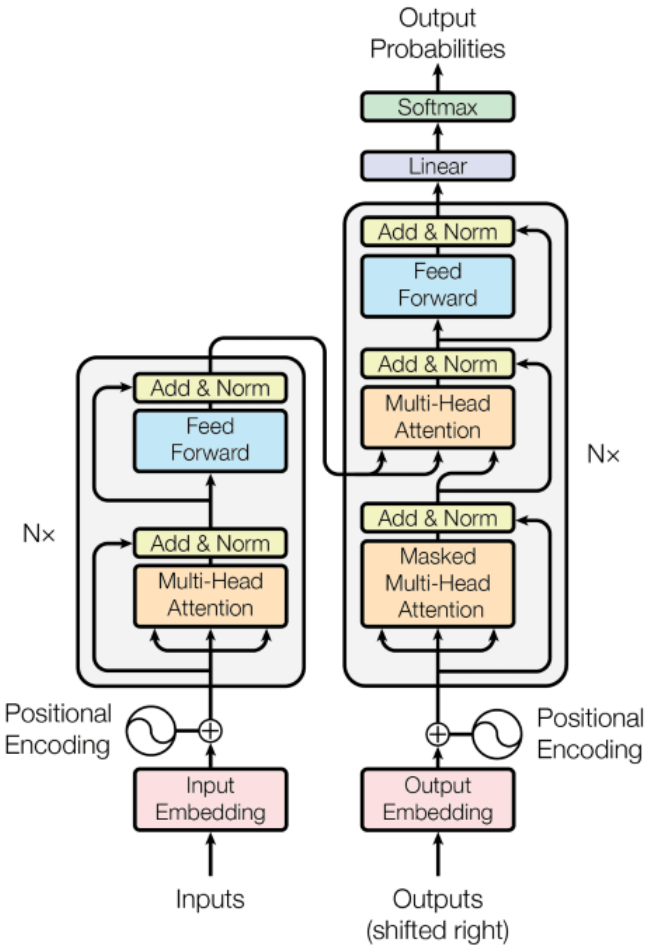

Figure 2.5: Architettura di un Transformer.

2.2.8 Utilizzazione

Un modello Transformer può essere impiegato per svolgere diversi compiti, come:

- **Natural language processing**: classificazione del testo, named entity recognition, question answering, modellazione del linguaggio, riepilogo, traduzione, scelta multipla, e generazione di testo.
- **Computer Vision**: classificazione di immagini, riconoscimento di oggetti, segmentazione.
- **Audio**: riconoscimento automatico di un discorso e classificazione di audio.

- **Multimodale**: riconoscimento ottico dei caratteri, estrazione di informazioni da documenti scansionati, classificazione video.

Per poter utilizzare un modello Transformer è necessario, inanzitutto, scaricare il relativo pacchetto:

```
pip install transformers
```

Utilizare Transformers, per esempio, per l'attività di NER, è necessario selezionare un modello pre allenato ed un tokenizer per poter creare una pipeline. Infine passando una sequenza testuale alla pipeline vengono estratte le entità riconosciute. L'algoritmo quindi sarebbe il seguente:

```
from transformers import pipeline,
AutoTokenizer, AutoModelForTokenClassification

tokenizer = AutoTokenizer.from_pretrained
        ("emilyalsentzer/Bio_ClinicalBERT")

model = AutoModelForTokenClassification.from_pretrained
        ("emilyalsentzer/Bio_ClinicalBERT")

ner_pipe = pipeline("ner", model=model, tokenizer=tokenizer)
sequence = "This expression of NT-3 i supporting cells
in embryos and neonates may even preserve in Brn3c
null mutants the numerous spiral sensory neurons in
the apex of 8-day old animals."

for entity in ner_pipe(sequence):
    print(entity)
```

2.3 BERT

Bidirectional encoder representations per Transformers, denominato BERT, è un nuovo modello di rappresentazione del linguaggio per Transformer. Questo modello è pre-addestrato su un ampio corpus di dati in lingua inglese in modalità auto-supervisionata, ciò significa che è stato preaddestrato solo sui testi grezzi, senza che gli esseri umani li etichettassero in alcun modo (motivo per cui può utilizzare molti dati disponibili pubblicamente) con un processo automatico per generare input ed etichette da quei testi. BERT ha ottenuto risultati all'avanguardia in diversi campi grazie alla sua struttura semplice ma empiricamente potente.

2.3.1 Panoramica

BERT utilizza Transformer per generare un modello linguistico, utilizzando solo il meccanismo del codificatore. A differenza dei modelli direzionali che leggono l'input di un testo in sequenza, i modelli BERT sono bidirezionali, ovvero osservano le parole circostanti (sia sinistra che destra) per comprendere il contesto. I modelli sono pre addestrati su enormi volumi di testo per apprenderne le relazioni ed ottenendo così un vantaggio rispetto ad altre tecniche.

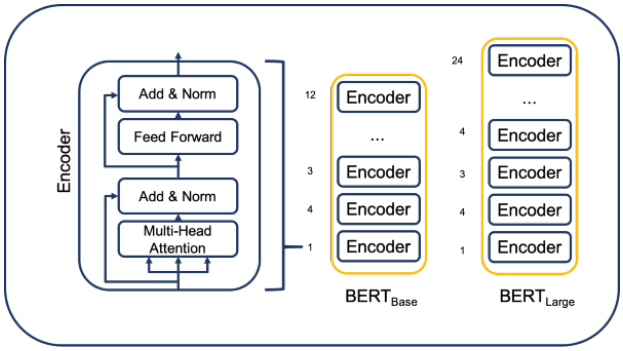

Figure 2.6: Struttura del modello BERT formato solo dal codificatore del Transformer.

2.3.2 Struttura del modello

BERT è fondamentalmente uno stack encoder dell'architettura Transformer, introdotta nell'articolo [17]. Un Transformer, di base, è costituito da un codificatore, utilizzato per leggere l'input di testo, e un decodificatore, finalizzato a produrre una previsione per l'attività. Poiché l'obiettivo di BERT è generare un modello di rappresentazione del linguaggio, necessita solo del meccanismo del codificatore che genera una rappresentazione z_i di una parola. La struttura di BERT è graficamente visibile consultando la figura 2.6. Il numero di strati, denominato **L**, la grandezza degli strati nascosti , denominata **H**, e il numero di teste di self-attention, denominate **A**, sono le caratteristiche necessarie per la definizione dei due principali modelli :

- $BERT_{base}$. Ugual numero di blocchi del Transformer e minor dimensione degli strati nascosti, L=12, H=768, A=12, Numero totale di parametri = 110M.
- $BERT_{large}$. Enorme rete con il doppio dei livelli di attenzione rispetto a $BERT_{base}$, raggiunge risultati all'avanguardia nelle attività di NLP, L=24, H=1024, A=16, Numero totale di parametri = 340M.

2.3.3 Input ed Output

Ogni sequenza di input passata al modello viene vista come una sequenza di token, che vengono convertiti in vettori e poi elaborati nella rete neurale. Prima che l'elaborazione possa iniziare, BERT ha bisogno che l'input venga manipolato e decorato con alcuni metadati extra:

- **Token embeddings**. La sequenza di input viene suddivisa in token utilizzando un dizionario di 300000 token, il WordPiece Tokens. Un token speciale di classificazione, CLS, viene aggiunto all'inizio di ogni sequenza ed un token di separazione, SEP, viene inserito alla fine di ogni frase.
- **Segment embeddings**. Ipotizzando di avere due frasi, A e B, un marcatore viene aggiunto a ciascun token della sequenza per indicare l'appartenenza alla frase A o alla frase B. Ciò consente al codificatore di distinguere tra le frasi.
- **Positional embeddings**. Un incorporamento posizionale viene aggiunto a ciascun token per indicare la sua posizione nella frase.

Una visualizzazione grafica della sequenza di manipolazioni a cui è sottoposto l'input da parte del modello BERT può essere trovata all'immagine 2.7.
L'output è una sequenza di vettori in cui ogni vettore corrisponde ad un

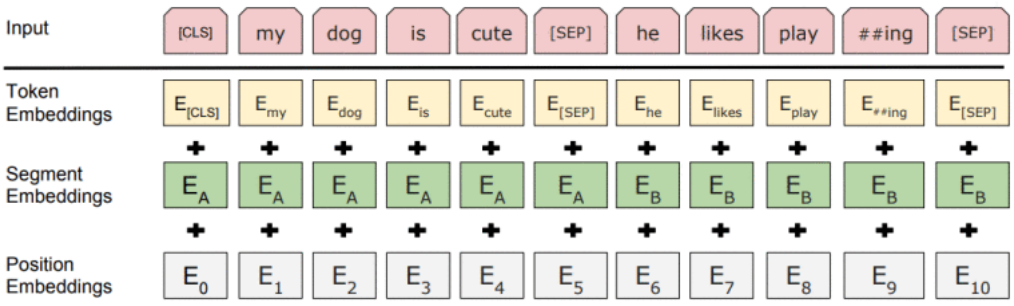

Figure 2.7: Sequenza di manipolazioni che deve subire una frase di input.

token di input con lo stesso indice. Per una determinata sequenza di token, la rappresentazione è quindi costruita sommando i precedenti tre embeddings.

2.3.4 Architettura Fine-Tuning

Contrariamente ai modelli direzionali, BERT legge l'intera sequenza di parole in una sola volta. Il codificatore è pre-allenato per rappresentare una sequenza di input e capirne il contesto ed il significato. Le prestazioni all'avanguardia di

BERT si basano su un procedimento chiamato fine-tuning composto da due fasi:

- **Pre-training**. Questa fase viene effettuata su un set di dati non etichettato, pertanto non è di natura supervisionata, e si compone di due compiti:
  - **Masked LM (MLM)**. L'idea è semplice, il 15% delle parole in ciascuna sequenza viene mascherato sostituendolo con un token, MASK. Il modello tenta quindi di prevedere il valore originale delle parole mascherate, in base al contesto fornito dalle altre parole non mascherate nella sequenza. In termini tecnici, la previsione delle parole di output richiede:
    * **1**. Aggiunta di uno strato di classificazione al di sopra dell'ouput di codifica.
    * **2**. Moltiplicazione degli output dei vettori per la matrice degli incorporamenti, trasformandoli nella dimensione del vocabolario.
    * **3**. Previsione della parola nascosta tramite lo strato di Softmax, utilizzando una distribuzione della probabilità.

    Tuttavia, così facendo, il token MASK non viene visualizzato durate la fase di fine-tuning. Per evitare questo problema l'80% dei token viene rimpiazzato con una maschera, il 10% sostituito con un token casuale e il restante 10% viene lasciato immutato.
  - **Next Sentence Prediction (NSP)**. Per comprendere la relazione tra due frasi, BERT utilizza anche la previsione della frase successiva. Un modello pre-addestrato con questo tipo di comprensione è rilevante per attività come la risposta alle domande. Durante l'addestramento il modello riceve come input coppie di frasi e impara a prevedere se la seconda frase è anche la frase successiva nel testo originale. Come abbiamo visto in precedenza, BERT separa le frasi con uno speciale token [SEP]. Durante l'addestramento il modello viene alimentato con due frasi di input alla volta in modo tale che il 50% delle volte la seconda frase arriva dopo la prima ed il 50% delle volte è una frase casuale dall'intero corpus. A BERT viene quindi richiesto di prevedere se la seconda frase è casuale o meno, assumendo che la frase casuale sarà disconnessa dalla prima frase.
- **Fine-tuning**. Il modello viene addestrato utilizzando dati di addestramento supervisionati da altre attività. I parametri del codificatore sono inizializzati utilizzando i parametri provenienti dalla fase precedente di pre-training. BERT può essere utilizzato per un'ampia varietà di attività linguistiche aggiungendo solo un piccolo livello al modello principale.

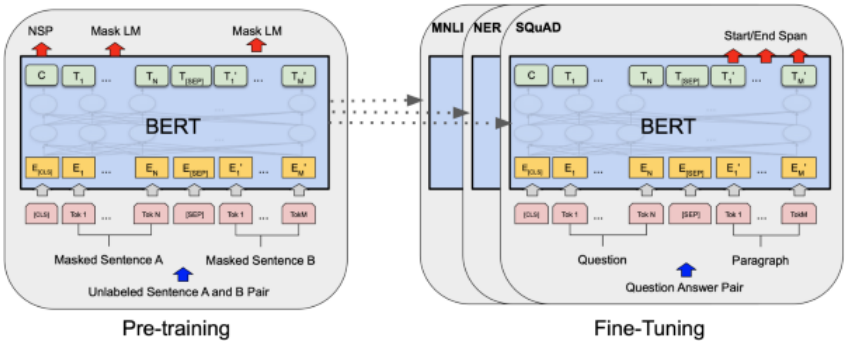

Figure 2.8: Pre training e fine tuning svolto dal modello BERT

La rappresentazione delle fasi di pre-training e fine-tuning è disponibile presso l'immagine 2.8.

2.3.5 Performance

Il modello BERT consente di ottenere un'efficienza superiore a tutti i modelli utilizzati in precedenza nel campo del Natural Language Processing. In seguito vengono presentati i risultati rispetto a diversi compiti nel campo del NLP:

- **General Language Understanding Evaluation (GLUE)**. L'attività di valutazione della comprensione generale del linguaggio, spiegata nell'articolo [18], è una raccolta di diverse attività di comprensione del linguaggio naturale. Questi includono MNLI (Multi-Genre Natural Language Inference), QQP (Quora Question Pairs), QNLI (Question Natural Language Inference), SST-2 (The Stanford Sentiment Treebank), CoLA (Corpus of Linguistic Acceptability) ecc. Entrambi, BERTBASE e BERTLARGE superano di gran lunga i modelli precedenti, rispettivamente del 4,5% e del 7%. I risultati ottenuti nell'attività di GLUE sono consultabili all'immagine 2.9.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$BERT_{BASE}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$BERT_{LARGE}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Figure 2.9: Risultato ottenuto da BERT nell'attività di General Language Understanding Evaluation.

- **Standford Question Answering(SQuAD)**. Il Question Answer Dataset di Standford è una raccolta di 100.000 coppie di domande e risposte di crowd source. Data una frase presa da wikipedia ed una domanda il compito è prevedere l'estensione del testo della risposta nel passaggio. Il BERT con le migliori prestazioni ha un punteggio F1 pari a 93.2 superiore di 2 punti rispetto al precedente ottenuto dagli umani. .
- **Situations With Adversarial Generations(SWAG)**. Il set di dati SWAG contiene 113.000 attività di completamento di frasi che valutano la risposta più adatta utilizzando un'inferenza basata sul buon senso. Data una frase, il compito è scegliere la continuazione più plausibile tra quattro scelte. $BERT_large$ supera l'OpenAI GPT, il modello migliore per questa attività, dell'8,3%. Funziona anche meglio di un essere umano esperto. I risultati ottenuti con SWAG sono consultabili all'immagine 2.10.

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
OpenAI GPT	-	78.0
$\text{BERT}_{\text{BASE}}$	81.6	-
$\text{BERT}_{\text{LARGE}}$	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0

Figure 2.10: Risultato ottenuto da BERT sul dataset SWAG.

Concludendo si può notare che BERT è stato in grado di migliorare l'accuratezza, o punteggio F1, in molte attività di elaborazione del linguaggio naturale.

2.3.6 Utilizzazione

BERT è facilmente utilizzabile e scaricabile seguendo il link:
https://github.com/google-research/bert

2.4 BioBERT

Il text mining biomedico assume sempre più importanza data la rapida crescita del numero di documenti biomedici presenti. Con i progressi nell'elaborazione del linguaggio naturale, l'estrazione di informazioni preziose dalla

letteratura biomedica ha guadagnato popolarità tra i ricercatori e il deep learning ha favorito lo sviluppo di efficaci modelli di text mining biomedico. Tuttavia, poiché i modelli di deep learning richiedono una grande quantità di dati di addestramento, il text mining biomedico con il deep learning spesso fallisce a causa delle dimensioni ridotte dei set di dati di addestramento attualmente disponibili per il campo biomedico. Viene introdotto BioBERT (Bidirectional Encoder Representations from Transformers for Biomedical Text Mining), che è un modello di rappresentazione del linguaggio specifico del dominio biomedico e pre-addestrato su corpus biomedici. Basato sull'architettura BERT, BioBERT trasferisce efficacemente la conoscenza di una grande quantità di testi biomedici in modelli di text mining biomedici.

2.4.1 Architettura

BioBERT ha sostanzialmente la stessa struttura di BERT, le differenze sono graficamente visibili all'immagine 2.11, e segue lo stesso procedimento fine-tuning composto da due fasi:

- **Pre-training**. Essendo un modello di rappresentazione linguistica generica, BERT è stato pre-addestrato su Wikipedia, in lingua inglese, e BooksCorpus[19]. Tuttavia, i testi dei domini biomedici contengono un numero considerevole di nomi propri e termini specifici del dominio. Di conseguenza i modelli NLP progettati per scopi generali ottengono scarse prestazioni in attività di text mining biomedico. BioBERT nasce per migliorare le prestazioni ottenute in precedenza in domini biomedici, quindi il corpus utilizzato per il pre-addestramento comprende riassunti di **PubMed** e articoli full-text di **PubMed Central**. Per la tokenizzazione delle sequenze di input, BioBERT, utilizza WordPiece dove ogni nuova parola può essere rappresentata da parole secondarie frequenti. Non è stato costruito un vocabolario basato su corpus biomedici, ma è stato utilizzato lo stesso di BERT per garantire la compatibilità tra i due modelli, il riutilizzo di modelli BERT pre-addestrati su dominio generale e per garantire l'intercambialità tra modelli esitenti di BERT e BioBERT.

- **Fine tuning**. Proprio come BERT anche BioBERT può essere utilizzato per uno svariato numero di attività modificando i parametri del modello.

2.4.2 Performance

Le prestazioni ottenute da BERT vengono significativamente superate da quelle di BioBERT nelle seguenti tre attività rappresentative di text mining

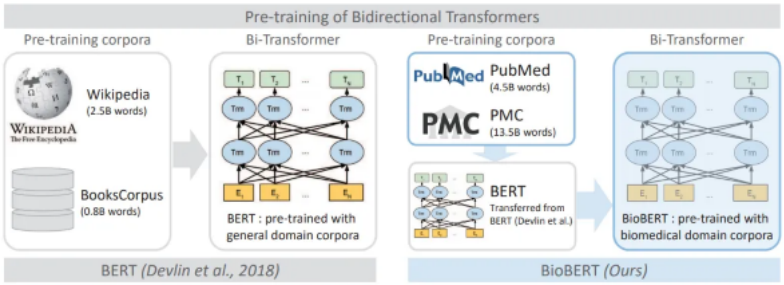

Figure 2.11: Differenza tra il corpus utilizzato da BERT e BioBERT nella fase di pre-training.

biomedico:

- **Named Entity Recognition**. BioBERT ha ottenuto un punteggio F1 più alto di 0,62 per l'estrazione di entità in testi biomedici. Questo risultato può essere verificato osservando l'immagine 2.12.

				BERT	BioBERT v1.0			BioBERT v1.1
Type	Datasets	Metrics	SOTA	(Wiki + Books)	(+ PubMed)	(+ PMC)	(+ PubMed + PMC)	(+ PubMed)
Disease	NCBI disease	P	88.30	84.12	86.76	86.16	**89.04**	88.22
		R	89.00	87.19	88.02	89.48	89.69	**91.25**
		F	88.60	85.63	87.38	87.79	89.36	**89.71**
	2010 i2b2/VA	P	87.44	84.04	85.37	85.55	**87.50**	86.93
		R	86.25	84.08	85.64	85.72	85.44	**86.53**
		F	**86.84**	84.06	85.51	85.64	86.46	86.73
	BC5CDR	P	**89.61**	81.97	85.80	84.67	85.86	86.47
		R	83.09	82.48	86.60	85.87	87.27	**87.84**
		F	86.23	82.41	86.20	85.27	86.56	**87.15**
Drug/chem.	BC5CDR	P	**94.26**	90.94	92.52	92.46	93.27	93.68
		R	92.38	91.38	92.76	92.63	**93.61**	93.26
		F	93.31	91.16	92.64	92.54	93.44	**93.47**
	BC4CHEMD	P	92.29	91.19	91.77	91.65	92.23	**92.80**
		R	90.01	88.92	90.77	90.30	90.61	**91.92**
		F	91.14	90.04	91.26	90.97	91.41	**92.36**
Gene/protein	BC2GM	P	81.81	81.17	81.72	82.86	**85.16**	84.32
		R	81.57	82.42	83.38	84.21	83.65	**85.12**
		F	81.69	81.79	82.54	83.53	84.40	**84.72**
	JNLPBA	P	**74.43**	69.57	71.11	71.17	72.68	72.24
		R	83.22	81.20	83.11	82.76	83.21	**83.56**
		F	**78.58**	74.94	76.65	76.53	77.59	77.49
Species	LINNAEUS	P	92.80	91.17	91.83	91.62	**93.84**	90.77
		R	**94.29**	84.30	84.72	85.48	86.11	85.83
		F	**93.54**	87.60	88.13	88.45	89.81	88.24
	Species-800	P	**74.34**	69.35	70.60	71.54	72.84	72.80
		R	75.96	74.05	75.75	74.71	**77.97**	75.36
		F	74.98	71.63	73.08	73.09	**75.31**	74.06

Figure 2.12: punteggio di BioBert nell'attività di NER

- **Relation Extraction**. Questa attività, anche chiamata RE, consiste nel classificare le relazioni di entità etichettate nell'ambito biomedico. BERT, nonostante le buone prestazioni, è stato superato da BioBERT con un punteggio F1 più alto di 2,8, come può essere visto dall'immagine 2.13.
- **Question Answering**. Il risultato ottenuto in questa attività è il migliore, BioBERT ottiene un punteggio F1, consultabile all'immagine 2.14, di 8 punti più alto rispetto ai modelli all'avanguardia di BERT.

				BERT	BioBERT v1.0			BioBERT v1.1
Relation	Datasets	Metrics	SOTA	(Wiki + Books)	(+ PubMed)	(+ PMC)	(+ PubMed + PMC)	(+ PubMed)
Gene–disease	GAD	P	**79.21**	74.28	76.43	75.20	75.95	77.32
		R	**89.25**	85.11	87.65	86.15	88.08	82.68
		F	**83.93**	79.29	81.61	80.24	81.52	79.83
	EU-ADR	P	76.43	75.45	78.04	**81.05**	80.92	77.86
		R	**98.01**	96.55	93.86	93.90	90.81	83.55
		F	85.34	84.62	84.44	**86.51**	84.83	79.74
Protein–chemical	CHEMPROT	P	74.80	76.02	76.05	**77.46**	75.20	77.02
		R	56.00	71.60	74.33	72.94	75.09	**75.90**
		F	64.10	73.74	75.18	75.13	75.14	**76.46**

Figure 2.13: punteggio di BioBert nell'attività di Relation extraction

			BERT	BioBERT v1.0			BioBERT v1.1
Datasets	Metrics	SOTA	(Wiki + Books)	(+ PubMed)	(+ PMC)	(+ PubMed + PMC)	(+ PubMed)
BioASQ 4b	S	20.01	27.33	25.47	26.09	**28.57**	27.95
	L	28.81	44.72	44.72	42.24	**47.82**	44.10
	M	23.52	33.77	33.28	32.42	**35.17**	34.72
BioASQ 5b	S	41.33	39.33	41.33	42.00	44.00	**46.00**
	L	56.67	52.67	55.33	54.67	56.67	**60.00**
	M	47.24	44.27	46.73	46.93	49.38	**51.64**
BioASQ 6b	S	24.22	33.54	**43.48**	41.61	40.37	42.86
	L	37.89	51.55	55.90	55.28	**57.77**	**57.77**
	M	27.84	40.88	48.11	47.02	47.48	**48.43**

Figure 2.14: punteggio di BioBert nell'attività di QA

2.4.3 Utilizzazione

Gli autori di BioBERT hanno rilasciato gratuitamente il codice per svolgere sia la fase di pre-training sia quella di fine tuning al seguente link:
https://github.com/dmis-lab/biobert

2.5 SciBERT

L'aumento esponenziale del volume delle pubblicazioni scientifiche negli ultimi decenni ha fatto della NLP uno strumento essenziale per la lettura automatica e la conoscenza di documenti scientifici. I recenti progressi nella NLP sono stati guidati dall'adozione di modelli neurali profondi, ma l'addestramento di tali modelli spesso richiede grandi quantità di dati etichettati. Ottenere questi dati etichettati di alta qualità nel settore scientifico risulta essere un'attività costosa ed impegnativa. Allo scopo di superare tale limite nasce SciBERT, ovvero, un modello linguistico pre-addestrato basato su BERT, utilizzato per affrontare la mancanza di dati etichettati di alta qualità in campo scientifico. L'utilizzo di SciBERT, introdotto nell'articolo [20], è stato testato in diversi compiti del Natural Language Processing, tra cui:

- **Named Entity Recognition**
- **PICO Extraction**
- **Text Classification**

- **Relation Classification**
- **Dependency Parsing**

In tutte queste attività SciBERT è risultato essere un modello più efficiente utilizzando dataset biomedici, le performance ottenute possono essere visualizzate osservando l'immagine 2.15: Vi sono due differenti versioni di SciBERT,

Task	Dataset	SOTA	BERT-Base	SciBERT	
				BaseVocab	SciVocab
NER	BC5CDR	87.12	85.72	88.11	**88.94**
	JNLPBA	78.58	74.48	75.83	**75.95**
	NCBI-disease	87.34	85.49	**86.91**	86.45
PICO	EBM-NLP	66.30	70.07	70.82	**71.18**
REL	ChemProt	64.10	69.22	73.7	**76.12**
CLS	PubMed 20k RCT	92.6	86.19	86.80	**86.81**
DEP	GENIA - LAS	91.92	91.29	91.26	**91.41**
	GENIA - UAS	92.84	92.33	92.32	**92.46**
NER	SciERC	64.20	62.95	65.12	**65.5**
REL	SciERC	n/a	72.46	74.42	**74.64**
CLS	ACL-ARC	53.0	60.68	**65.79**	65.71
CLS	Paper Field	n/a	63.39	64.02	**64.07**
	ScienceCite	84.0	84.43	84.43	**84.99**
			75.53	77.27	**77.64**

Figure 2.15: Performance di SciBERT in ambito scientifico.

ovvero:

- **Cased**. Il testo è uguale al testo di input, ovvero non viene effettuata nessuna modifica.
- **Uncased**. Il testo è stato scritto in minuscolo prima della fase di tokenizzazione di WordPiece.

2.5.1 Differenze con BERT

BERT utilizza WordPiece per la tokenizzazione non supervisionata del testo di input. WordPiece, si basa su un vocabolario di unità di sottoparole. Il vocabolario è costruito in modo tale da contenere le parole o le sottoparole usate più di frequente. Si fa riferimento al vocabolario originale rilasciato con Bert come BaseVocab. Ovviamente le parole e le sottoparole frequentemente

osservate nel testo scientifico differiscono da quelle che ricorrono nel testo di dominio generale. A tal fine per modellare meglio il testo scientifico viene costruito un nuovo vocabolario ovvero SciVocab, basato sul corpus scientifico. Il vocabolario costruito ha una dimensione impostata a 30K, simile a BERT. Per quanto concerne al Corpus, SciBert è stato allenato su un insieme di 1,14 Milioni di documenti di cui il 18% proveniente dal dominio dell'informatica e l'82% dal vasto dominio biomedico. In fase di allenamento è stato utilizzato il testo completo dei documenti e non solo il loro abstract in modo tale da ottenere un modello robusto in grado di far fronte al problema del rumore.

2.5.2 Utilizzazione

Il link per il download e l'utilizzo del modello SciBERT è:
https://github.com/allenai/scibert

2.6 Deep Metric Learning

Uno degli aspetti più sorprendenti del sistema visivo umano è la capacità di riconoscere oggetti e scene simili. Un essere umano non ha bisogno di centinaia di foto dello stesso volto per poterlo differenziare tra migliaia di altri e non ha bisogno di migliaia di immagini della Torre Eiffel per riconoscere quel punto di riferimento architettonico quando visita Parigi. Il Deep Metric Learning essenzialmente, tenta di capire quali oggetti sono visivamente simili e quali no combinando **Deep learning** e **metric learning**.

2.6.1 Metric Learning

Molti approcci in machine learning richiedono una misura il più accurata possibile della distanza tra due elementi. Nasce per questo il Metric learning un'attività che mira a costruire nuove metriche di distanza per compiti specifici analizzandone i dati. Tramite la metrica costruita il Metric Learning è in grado di avvicinare elementi simili e aumentare il divario tra elementi dissimili. I casi d'uso sono diversi:

- **Nearest neighbors models**. La metrica imparata può essere utilizzata per migliorare diversi modelli di nearest neighbors utilizzati per la classificazione, la regressione, la detenzione di anomalie ecc...

- **Clustering**. Il metric learning fornisce un modo per orientare i cluster trovati da algoritmi come K-Means verso la semantica prevista.

- **Information retrieval**. La metrica appresa può essere utilizzata per recuperare gli elementi di un database che sono semanticamente più vicini a un elemento di query.

- **Riduzione delle dimensionalità**. Il metric learning può essere visto come un modo per ridurre la dimensione dei dati in un ambiente (debolmente) supervisionato.

Il meatric learning fornisce una nuova rappresentazione dei dati, come mostrato nell'articolo [21], utilizzando la relazione di somiglianza tra i campioni, che ha una discriminazione più significativa e potente.

2.6.2 Mahalanobis Distances

La maggior parte degli algortimi di metric learning implementati fino ad oggi apprendono le cosiddette distanze di Mahalanobis. Preso un insieme di campioni di allenamento $X = [x_1, x_2, ..., x_N] \in R^{dxN}$, dove $x_i \in R^d$ è l'i-esimo esempio di allenamento ed N è il numero totale di campioni di addestramento, la distanza tra x_i ed x_j viene così definita:

$$d_M(x_i, x_j) = \sqrt{(x_i - x_j)^T M(x_i - x_j)} \tag{2.14}$$

$d_M(x_i, x_j)$ corrisponde ad una metrica della distanza e gode della proprietà di non negatività, simmetria e disuguaglianza triangolare, che possono essere sfruttate per decomporre M, M=W^TW, e riformulare la metrica come segue:

$$d_M(x_i, x_j) = ||Wx_i - Wx_2||_2 \tag{2.15}$$

Come si può vedere dall'equazione, W ha una proprietà di trasformazione lineare. Grazie a questa proprietà si può esprimere la distanza euclidea nello spazio trasformato come la distanza di Mahalanobis nello spazio originale per due campioni.

2.6.3 Deep Learning

Il deep learning è un sottocampo dell'apprendimento automatico che prevede l'addestramento di reti neurali artificiali per apprendere modelli complessi nei dati. Queste reti neurali sono composte da più strati di "neuroni" interconnessi, che elaborano l'input dati e imparano a fare previsioni o decisioni. Il termine "profondo" si riferisce all'uso di più livelli nella rete neurale a differenza dei tradizionali algoritmi di apprendimento automatico, che in genere utilizzano un singolo livello decisionale. Un vantaggio chiave degli algoritmi di deep learning

è la loro capacità di apprendere le funzionalità da dati grezzi in modo tale da apprendere automaticamente rappresentazioni complesse e astratte dei dati che possono essere altamente efficaci per attività come il riconoscimento di immagini e parole. Gli algoritmi di deep learning hanno anche la capacità di apprendere gerarchicamente. Tuttavia, anche gli algoritmi di deep learning presentano alcune limitazioni, essi possono richiedere un grande quantità di dati di addestramento etichettati e possono essere computazionalmente costosi da addestrare. Inoltre questi algoritmi possono anche essere difficili da interpretare, dato che l'interno processo decisionale della rete neurale non è trasparente. Nel complesso il deep learning rimane un potente strumento per apprendere schemi complessi nei dati. I risultati raggiunti sono impressionanti su una vasta gamma di applicazioni, compreso il riconoscimento di immagini e parole, l'elaborazione del linguaggio naturale, come dimostrato nell'articolo [22] e la traduzione del linguaggio.

2.6.4 Funzionamento Deep Metric Learning

Il deep metric learning, come spiegato nell'articolo [23], è un sottocampo del machine learning che prevede l'apprendimento di metriche di distanza per il confronto e l'analisi di punti dati posizionati in spazi ad alta dimensione. Queste metriche di distanza possono essere utilizzate per definire la somiglianza tra punti dati. L'obiettivo del deep metric learning è, quindi, apprendere una metrica di distanza che rifletta la sottostante struttura dei dati, come le relazioni tra diverse classi o la somiglianza tra le immagini. Ciò si ottiene in genere addestrando una rete neurale profonda per mappare i dati in uno spazio di incorporamento di dimensioni inferiori, dove la distanza tra i punti riflette la loro somiglianza. La rete viene addestrata utilizzando una funzione di loss che incoraggia la rete a mappare dati simili in posizioni vicine e punti dissimili in posizioni molto distanti nello spazio di incorporamento. Il deep metric learning è stato applicato ad un'ampia gamma di attività, incluso il recupero di immagini, il riconoscimento di oggetti e la classificazione del testo ottenendo prestazioni migliori rispetto ad approcci tradizionali. Ad esempio, nella classificazione del testo, gli algoritmi di deep metric learning possono essere utilizzati per apprendere una metrica della distanza che rifletta la somiglianza semantica tra i testi, consentendo una precisa classificazione dei testi in diverse categorie. Sostanzialmente, per un set di punti dati X e con le rispettive etichette Y (un set finito e discreto), l'obbiettivo è quello di allenare un modello neurale incorporato f_θ(.):X->R^n(dove con θ si intendono i pesi appresi) con una distanza D:R^n->R, in modo tale che per due punti qualsiasi x_1, $x_2 \in X$, con le rispettive etichette y_1, $y_2 \in Y$, la combinazione D($f_\theta(x_1)$, $f_\theta(x_2)$) produca valori piccoli se y_1, y_2 sono uguali e valori maggiori se non lo sono. Pertanto

risulta chiaro che il problema del Deep Metric Learning si riduce alla scelta dell'architettura della rete, denominata F_θ, e della funzione di loss, denominata L(θ).

2.6.5 Approcci contrastivi

L'idea alla base degli approcci contrastivi è quella di progettare una funzione di loss che riunisca direttamente i campioni di dati con la stessa etichetta(cioè campioni simili) e allontani i campioni dissimili, da cui il nome "Contrastivo". La funzione di distanza nello spazio di incorporamento per questi approcci è solitamente fissata come metrica l_2.

Contrastive Loss

Questa è una classica funzione di loss per l'apprendimento metrico. Contrastive loss è una delle più semplici e intuitive funzioni create fino ad oggi. Definendo x_1 e x_2 come alcuni campioni nel dataset di partenza e y_1, y_2 come le etichette corrispondenti, la funzione di loss viene espressa come segue:

$$L_{contrast} = 1_{y_1=y_2} D^2_{f\theta}(x_1, x_2) + 1_{y_1!=y_2} max(0, \alpha - D^2_{f\theta}(x_1, x_2)) \tag{2.16}$$

Dove α è il margine. La ragione del perché sia necessario un valore di margine è che altrimenti la rete F_θ imparerebbe a "barare" mappando tutte le X sullo stesso punto e ponendo le distanze tra tutti i punti uguali a 0.

Triplet Loss

Triplet Loss è considerata di gran lunga la funzione di loss più popolare e ampiamente utilizzata per l'apprendimento metrico. Prendendo x_a, x_p, x_n come campioni da un dataset e le rispettive etichette y_a, y_p, y_n, in modo tale che $y_a = y_p$ e $y_a != y_n$, si può definire la triplet loss:

$$L_{triplet} = max(0, D^2_{f\theta}(x_a, x_p) - D^2_{f\theta}(x_a, x_n) + \alpha) \tag{2.17}$$

dove α è il margine per scoraggiare la rete F_θ a mappare l'intero set di dati X nello stesso punto. Solitamente x_a è chiamato ancora, x_p è chiamato campione positivo, poiché ha la stessa etichetta di x_a, e x_n è chiamato campione negativo, perché ha un etichetta differente dalle altre. Il funzionamento della triplet loss può essere graficamente visualizzato all'immagine 2.16.

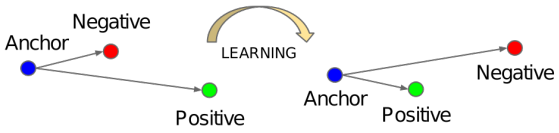

Figure 2.16: Funzionamento della triplet loss.

Multi similarity Loss

La multi similarity loss implementa uno schema composto da due passaggi iterativi, ovvero mining e ponderazione, concetto suggerito nell'articolo [24]. Sostanzialmente, le coppie vengono prima campionate misurandone la somiglianza e vengono selezionate le coppie informative, le quali vengono ulteriormente ponderate utilizzando l'auto somiglianza e la somiglianza negativa. Entrando nei particolari le due fasi sono:

- **Pair minig**. Per prima cosa vengono selezionate le coppie informative calcolando Similarity-P, che misura la somiglianza relativa tra coppie negative e positive che hanno lo stesso ancoraggio. Nello specifico, una coppia negativa viene confrontata con la coppia positiva più dura (con la somiglianza più bassa), mentre una coppia positiva viene campionata confrontandola con una negativa con la somiglianza più grande. Formalmente si assume x_i come ancora, una coppia negativa x_i, x_j viene selezionata se S_ij soddisfa la condizione:

$$S_{-ij} > min_{yk=y_i} S_{ik} - \in \tag{2.18}$$

$$dove \in ilmarginepreimpostato. \tag{2.19}$$

  Una coppia positiva x_i, x_j viene invece selezionata se S_ij soddisfa la condizione:

$$S_{+ij} < max_{yk!=y_i} S_{ik} + \in \tag{2.20}$$

  Per un ancora x_i l'insieme di indici sulle coppie positive viene indicato come P_i mentre quello sulle coppie negative come N_i.

- **Pair Weighting**. Tramite il Pair mining vengono selezionate le coppie più informative e vengono scartate quelle meno informative. Per una coppia negativa x_i, x_j E N_i, il suo peso w_ij può essere calcolato come:

$$w_{ij} = \frac{e^{\beta(S_{ij}-\lambda)}}{1 + \sum_{kappartienesimboloN_i} e^{\beta}(S_{ij} - \lambda)} \tag{2.21}$$

  Il peso w_ij per una coppia positiva x_i, x_j E P_i viene invece calcolato come segue:

$$w_{ij} = \frac{1}{e^{-\alpha(\lambda - S_{ij})} + \sum_{kappartienesimboloP_i} e^{-\alpha(S_{ik} - S_{ij})}} \tag{2.22}$$

Integrando, quindi, le formule di pair mining e pair weighting in un unico framework, la multi similarity loss può essere così calcolata:

$$L_{MS} = \frac{1}{m}\sum_{i=1}\frac{1}{\alpha}\log[1+\sum_{kEP_i} e^{-\alpha(S_{ik}-\lambda)}] + \frac{1}{\beta}\log[1+\sum_{kEN_i} e^{\beta(S_{ik}-\lambda)}] \quad (2.23)$$

2.7 Information Retrieval

L'Information retrieval, denominato IR, è definito come un programma software che si occupa della rappresentazione, memorizzazione, organizzazione e recupero dell'informazione testuale da un archivio di documenti. In particolare data una collezione di documenti e una richiesta di informazioni da parte dell'utente lo scopo di un sistema IR è quello di trovare informazioni che potrebbero essere utili o rilevanti per l'utente. Le ricerche possono essere basate su documenti originali full-text o su un'indicizzazione basata sul contenuto. Il sistema IR aiuta gli utenti a trovare le informazioni inerenti alla richiesta ricevuta ma non restituisce esplicitamente le risposte alla domanda. Il recupero delle informazioni, come spiegato nell'articolo [25], è fondamentale per dare un senso ai dati. Gli IR indicizzano i dati consentendo agli utenti di accedere online a informazioni di cui non potrebbero, altrimenti, essere a conoscenza.

2.7.1 Componenti di un sistema di IR

Si ritiene che il recupero delle informazioni sia la forma dominante di accesso alle informazioni. Il sistema IR aiuta gli utenti a trovare le informazioni di cui hanno bisogno ma non restituisce esplicitamente le risposte alla domanda, ma notificano in merito all'esistenza e all'ubicazione di documenti che potrebbero contenere le informazioni richieste. Le fasi che compongono un sistema di IR sono:

- **Acquisizione**. In questa fase avviene la selezione di documenti e altri oggetti da varie risorse Web che sono costituite da documenti basati su testo. I dati richiesti vengono raccolti dal web e archiviati nel database.
- **Rappesentazione dei dati**. In questa fase i dati raccolti vengono strutturati e elaborati.
  - **Inverted Index**. La struttura dati primaria della maggior parte dei sistemi IR è sotto forma di indice invertito. Possiamo definire un indice invertito come una struttura dati che elenca, per ogni parola, tutti i documenti che la contengono e la frequenza delle occorrenze della parola stessa all'interno del documento. Semplifica la ricerca di "risultati" di una parola di ricerca.

  - **Stop word elimination**. Le stop word sono quelle parole ad alta frequenza che si ritiene improbabile siano utili per la ricerca. Hanno meno pesi semantici. Tutti questi tipi di parole sono in un elenco chiamato stop list. Ad esempio, gli articoli “a”, “an”, “the” e le preposizioni come “in”, “of”, “for”, “at” ecc. sono esempi di stop words. La dimensione dell'indice invertito può essere notevolmente ridotta dall'elenco di stop.

  - **Stemming**. Lo stemming, la forma semplificata dell'analisi morfologica, è il processo euristico di estrazione della forma base delle parole tagliando le estremità delle parole. Ad esempio, le parole ridere, ridere, ridere verrebbero derivate dalla radice della parola ridere.

- **Organizzazione dei file**. Esistono due tipi di metodi di organizzazione dei file, ovvero **sequenziale**, che contiene documenti per dati del documento, e **invertito**, il quale contiene per ogni termine un elenco di record . Un ultima possibile soluzione consiste nella combinazione dei due metodi spiegati in precedenza.

- **Query**. Un processo IR inizia quando un utente inserisce una query nel sistema. Le query sono dichiarazioni formali di informazioni necessarie, ad esempio stringhe di ricerca nei motori di ricerca web. Nel recupero delle informazioni, una query non identifica in modo univoco un singolo oggetto nella raccolta. Invece, diversi oggetti possono corrispondere alla query con diversi gradi di pertinenza.

I nuovi modelli utilizzati per il recupero delle informazioni (IR) utilizzano reti neurali profonde che possono apprendere rappresentazioni di testo o utilizzare incorporamenti pre-addestrati. Le DNN più utilizzate per IR sono, come spiegato nell'articolo [26], *Siamese network*, *Interaction-based networks*, reti neurali per classificare i risultati di ricerca in risposta a una query Il **document retrieval** è definito come un ramo dell'information retrieval che si occupa del recupero di documenti rilevanti rispetto ad una determinata richiesta fatta dall'utente. Questi documenti potrebbero essere qualsiasi tipo di testo principalmente non strutturato, come articoli di giornale, documenti immobiliari o paragrafi di un manuale.

2.8 Dense Passage Retrival

2.8.1 Question Answering

I sistemi di question answering, denominati QA, ([27]) di dominio aperto si occupano di produrre la risposta a domande di input utilizzando un'ampia raccolta di documenti. Questi sistemi hanno una struttura composta in due fasi:

- **Context Retriever**. Un context retriever seleziona un piccolo sottoinsieme di passaggi rilevanti per la domanda di input, che possono contenere risposte.

- **Lettore automatico**. Prende i contesti recuperati al passaggio precedente e si occupa dell'identificazione della risposta corretta.

Generalmente la fase di recupero viene implementata utilizzando meccanismi come **TF-IDF** o **BM25**, ovvero modelli di spazio vettoriale sparso. È stato dimostrato (citazione) che il meccanismo di recupero di informazioni può essere facilmente implementato utilizzando solo rappresentazioni dense, in cui gli incorporamenti vengono appresi da un piccolo numero di domande e passaggi da un semplice framework con doppio codificatore. Il sistema appena presentato è chiamato Dense Passage Retriever (DPR). Il DPR non solo supera le prestazioni sia di TF-IDF e BM25, ma introduce anche un miglioramento sull'accuratezza dell'attività di question answering end-to-end. Attraverso una serie di attenti studi di ablazione, la soluzione finale consiste nel confrontare tutte le coppie di domande e passaggi in un batch. Il Dense Passage Retriever (DPR) è eccezionalmente forte, poichè non solo supera BM25, ma si traduce anche in un sostanziale miglioramento sull'accuratezza del QA end-to-end.

2.8.2 Panoramica

Il Dense passage retrieval è un insieme di strumenti e modelli all'avanguardia per la componente di context retriever nell'attività di QA a dominio aperto. Data una collezione di M passaggi di testo, l'obbiettivo del DPR è di indicizzare tutti i passaggi in uno spazio continuo, tale da poter recuperare efficacemente i migliori k passaggi rilevanti rispetto alla domanda di input. I k passaggi selezionati sono quelli sui quali eseguire la lettura automatica. La dimensione di k dipende da una coppia di fattori come il ritardo che ci si aspetta nella pipeline, la recall, la disponibilità di calcolo, ecc., ma in generale qualsiasi valore di k compreso tra 10 e 50 risulta essere utile alle finalità del DPR.

2.8.3 Struttura di un DPR

Il DPR è un modello neurale per il recupero di informazioni che codifica la query q e il passaggio p in rappresentazioni vettoriali dense:

$$v_q = E_Q(q) \tag{2.24}$$

$$v_p = E_P(p) \tag{2.25}$$

E_Q ed E_P sono modelli BERT che producono in output una lista di vettori densi (h_1, ...,h_n) per ogni token preso dalla domanda di input. E_Q ed E_P sono inizialmente inizializzati con lo stesso valore per poi essere aggiornati indipendentemente nella fase di training. Il codifcatore Eq, inoltre, ha il compito di recuperare i k passaggi dei vettori più vicini al vettore della query. È possibile definire la similarità tra la domanda, o la query, q ed il passaggio p tramite la seguente funzione:

$$sim(q, p) = v_q^T v_p \tag{2.26}$$

Maggiore sarà il valore di similarità ottenuto più importante sarà il passaggio considerato.

Fase di Training di un DPR

L'obbiettivo è di creare uno spazio vettoriale dove le coppie di domanda e passaggio rilevanti avranno una distanza inferiore rispetto alle coppie irrilevanti. Ciò è possibile grazie all'apprendimento di una miglior funzione di incorporamento. Per allenare il modello il dataset è rappresentato come D $=\sum_{i=1} < q_i, p_i^+, p_{i,1}^-, ..., p_{i,n}^- >$ dove q_i è l'i-esima domanda. Ogni domanda è abbinata al suo corrispondente esempio positivo e ad un numero n di esempi negativi. La funzione di loss come probabilità logaritmica negativa del passaggio positivo viene calcolata come segue:

$$L(q_i, p_i^+, p_{i,1}^-, ..., p_{i,n}^-) = -\log \frac{e^{sim(q_i, p_i^+)}}{e^{sim(q_i, p_i^+)} + \sum_{j=1}^{n} e^{sim(q_i, p_{i,j}^-)}} \tag{2.27}$$

Per la fase di recupero, spesso, gli esempi positivi sono disponibili esplicitamente, mentre gli esempi negativi devono essere selezionati da un insieme estremamente grande. La modalità di selezione per esempi negativi è un fattore decisivo per l'apprendimento di un codificatore di alta qualità. Ci sono tre tipi di generazione di esempi negativi:

- **Random**. Un qualsiasi passaggio casuale all'interno dell'intero insieme di passaggi selezionati.

- **BM25**. I passaggi migliori selezionati da BM25 che non devono necessariamente contenere la risposta alla domanda.

- **Gold**. passaggi positivi accoppiati con altre domande che appaiono nel training set.

2.8.4 Performance DPR

Attualmente DPR è il metodo di cotext retrieval migliore, esso ha superato ance BM25. Un DPR addestrato utilizzando solo 1000 esempi supera già nettamente BM25, ovviamente l'aggiunta di più esempi migliora ulteriormente la precisione ottenuta con il DPR, ciò può essere osservato tramite l'immagine 2.17.

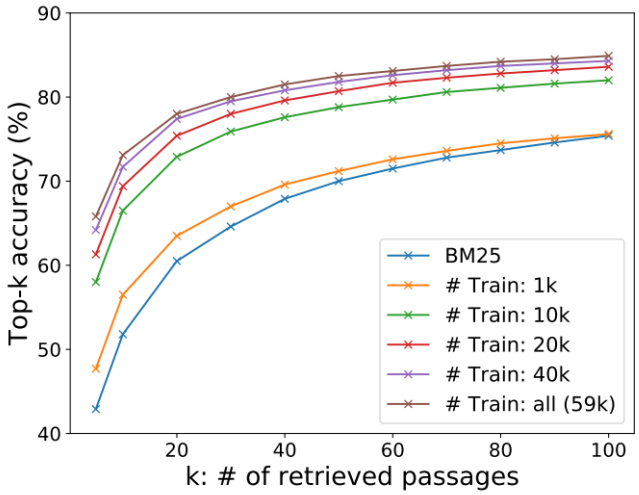

Figure 2.17: Performance di un DPR.

Sebbene DPR abbia prestazioni migliori rispetto a BM25 in generale, i due modelli sono qualitativamente differenti. Metodi come BM25 sono sensibili a parole chiave e frasi altamente selettive, ma non possono catturare bene le variazioni lessicali o le relazioni semantiche. Al contrario, DPR eccelle nella rappresentazione semantica, ma potrebbe non avere una capacità sufficiente per rappresentare frasi salienti che appaiono raramente. La risposta corretta alla prima domanda è restituita da DPR mentre BM25 restituisce una risposta irrilevante. Per quanto riguarda il secondo esempio, BM25 restituisce una risposta coerente mentre DPR, non riuscendo a catturare l'essenza della domanda, restituisce una risposta fuori contesto.

2.8.5 PolyDPR

Poly-Encoder

Proprio come DPR, anche il Poly Encoder utilizza due codificatori per codificare query e passaggi.La differenza è che la query è rappresentata da K vettori anziché da un singolo vettore, come accade nel DPR. Poly-Encoder presuppone che la query sia molto più lunga del passaggio considerato, che è in contrasto con il recupero delle informazioni e le attività di QA del dominio aperto nel dominio biomedico, dove i passaggi sono documenti lunghi e le query sono brevi e specifiche.

Struttura

PolyDPR, introdotto nell'articolo [28], nasce dall'integrazione del DPR e un poly encoder per utilizzare K vettori per rappresentare il contesto piuttosto che la query. Sostanzialmente vengono incluse K feature globali (m_1, ..., m_k) utilizzate per estrarre la rappresentazione vettoriale di tutti i token di un contesto.

$$v_c^i = \sum_n w_n^{m_i} h_n \tag{2.28}$$

$$dove(w_1^{m_i}, ..., w_n^{m_i}) = softmax(m_i^T h_1, ..., m_i^T h_n) \tag{2.29}$$

$$v_{c,q} = \sum_k w_k v_c^k \tag{2.30}$$

$$dove(w_1, ..., w_k) = softmax(v_q^T v_p^1, ..., v_q^T v_p^k) \tag{2.31}$$

Dove $v_{c,q}$ è una rappresentazione specifica del contesto della query viene calcolata utilizzando il meccanismo di attenzione. Per l'inferenza viene utilizzata una funzione alternativa per il calcolo delle somiglianze, questo a causa del fatto che durante l'inferenza risulta necessario il calcolo di una classifica del contesto dopo aver ottenuto tutte le rappresentazioni specifiche dei contesti per una query.

$$sim_{infer}(q, c) = max(v_q^T v_p^1, ..., v_q^T v_p^k) \tag{2.32}$$

Tramite questa funzione viene calcolato il punteggio di similarità per la query e ciascuna delle K rappresentazioni prendendo il massimo come punteggio di somiglianza tra query e contesto.

Capitolo 3

Tecnologie utilizzate

3.1 Clinical Knowledge Graph

Come ben noto, la promessa della medicina di precisione è quella di fornire un trattamento personalizzato basato sulla fisiologia unica di ciascun paziente e su un'analisi più completa dei fenotipi della malattia. Ciò richiede una perfetta integrazione di differenti dati, come dati clinici, di laboratorio, di imaging e multiomica (genomica, trascrittomica, proteomica o metabolomica). La quantità e la diversità dei dati biomedici a disposizione rende l'obbiettivo precedentemente anticipato molto difficile da raggiungere. Proprio per questo la comunità della ricerca biomedica ha da tempo riconosciuto la necessità di raccogliere, organizzare e strutturare i dati rilevanti, con conseguente adozione a livello comunitario di database biomedici. Questo tipo di organizzazione dei dati non è però sufficiente, sono necessarie soluzioni che integrino i diversi tipi di dati catturando le relazioni tra le entità molecolari e il fenotipo della malattia risultante. Per questo motivo è necessario un grafo della conoscenza che faciliti l'armonizzazione della proteomica con altri dati omici integrando al contempo i database biomedici pertinenti e il testo estratto dalle pubblicazioni scientifiche. Il **Clinical Knowledge Graph**[29], anche detto **CKG**, costituisce una prima soluzione, è una piattaforma open source progettata da Alberto Santos, Ana Rita Colaco, Annelaura Bach Nielsene Matthias Mann. Si tratta di una piattaforma composta, attualmente, da circa 20 milioni di nodi e 220 milioni di relazioni utilizzate per rappresentare dati sperimentali rilevanti, database pubblici e la loro letteratura. Il CKG è costruito su librerie scientifiche come Python, il che rende la piattaforma affidabile, facilmente mantenibile e in continuo miglioramento. La struttura del grafo, che può essere osservato graficamente alla figura 3.1, fornisce un modello di dati flessibile che è facilmente estendibile a nuovi nodi e alle corrispettive relazioni che diventano disponibili ai nuovi database. Il CKG incorpora gli ultimi algoritmi statistici e di apprendimento

automatico che accelerano drasticamente l'analisi e l'interpretazione dei flussi di lavoro.

Figure 3.1: Clinical Knowledge Graph con tutte le possibili etichette ed i tipi di relazione tra nodi.

3.1.1 Struttura

Il CKG include quattro diversi moduli funzionali:

- **analytic core**. Consente la formattazione e l'analisi dei dati di proteomica.
- **graphdb builder**. Si occupa della costruzione un database a grafo integrando i dati disponibili da una serie di database pubblicamente accessibili, esperimenti condotti dagli utenti, ontologie esistenti e pubblicazioni scientifiche.
- **graphdb connector**. Permette di connettersi e interrogare questo database grafico .

- **report manager**. Facilita la visualizzazione, il repository e l'analisi dei dati tramite report online e notebook Jupyter.

Questa architettura, mostrata nella figura 3.2, armonizza e integra perfettamente i dati e l'analisi fornita dall'utente. Inoltre, facilita la condivisione e la visualizzazione dei dati, nonché la loro l'interpretazione, generando risultati clinicamente rilevanti.

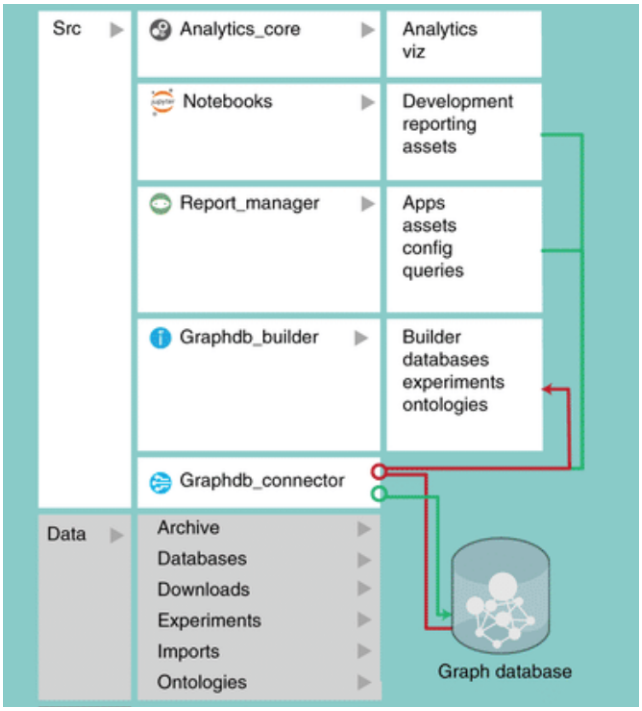

Figure 3.2: Visualizzazione dei quattro moduli che compongono un CKG.

Il core di analisi CKG implementa più algoritmi di data science aggiornati per l'analisi statistica e la preparazione, esplorazione, analisi e visualizzazione dei dati di proteomica. Il modello di dati del database del grafo CKG è stato progettato per integrare esperimenti di proteomica clinica multilivello e per annotarli con dati biomedici. Definisce diversi nodi (ad esempio, Proteina, Metabolita e Malattia) e i tipi di relazione tra di essi. Per scaricare il dataset per prima cosa è stata creata una directory in */mnt/ssd/barberini*, ovvero un disco di memoria esterno con molto spazio. Successivamente, una volta posizionati nella cartella creata in precedenza, si lancia il seguente comando su shell:

```
wget https://data.mendeley.com/datasets/mrcf7f4tc2
```

3.2 Neo4j

Neo4j, creato nel 2007, è un database a grafo no-SQL open source e ampliamente utilizzato in ambito commerciale. Neo4j è un database a grafo, ovvero un database composto da nodi(vertici) e relazioni(spigoli).I nodi rappresentano le entità, mentre le relazioni rappresentano l'associazione di questi nodi. Questo tipo di database è più adatto per alcune applicazioni di Big Data e analisi rispetto ai classici database composti da righe e colonne. Neo4j è l'unico database a grafo di alto livello che combina l'archiviazione nativa dei grafi, un'architettura scalabile ottimizzata per la velocità e la conformità ACID(atomicità, consistenza, isolazione e durabilità).

3.2.1 Database a grafo

In un tradizionale database relazionale o SQL, i dati sono organizzati in tabelle. Ogni tabella registra i dati in un formato specifico con un numero fisso di colonne, ciascuna colonna con il proprio tipo di dati (numero intero, ora/data, testo in formato libero, ecc.). Questo modello comporta dei limiti notevoli legati alla struttura rigida della tabella ed al fatto che ogni transazione legata a questi database comporti una serie di controlli e precauzioni che penalizzano le prestazioni del sistema[30]. Per questo motivo vengono progettati i database a grafo, utilizzati prettamente per l'archiviazione e la navigazione di relazioni tra dati. Graficamente la differenza tra un database a grafo ed uno relazionale è mostrata nella figura 3.3.

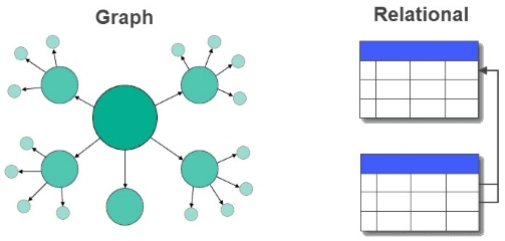

Figure 3.3: Semplificazione della struttura di neo4j in confronto alla struttura dei database SQL.

Fondamentalmente i database a grafo memorizzano i dati sotto forma di grafo, il quale permette di comprendere al meglio le connessioni e gli schemi all'interno delle informazioni. Un grafo è un concetto matematico utilizzato per descrivere una raccolta di nodi, proprietà e le relazioni etichettate. I database a grafo utilizzano nodi per identificare le entità e archi per individuare le relazioni tra le entità. Un arco è definito da un nodo iniziale ed uno finale,

dalla tipologia, dalla direzione Un nodo può avere infinite relazioni. I database a grafo presentano numerosi vantaggi, rispetto ai database relazionali, per casi d'uso come social network, motori di raccomandazione e rilevamento di frodi, dove è necessario creare relazioni tra dati ed eseguire rapidamente query su di esse. L'elaborazione rapida delle query sulle relazioni è dovuta al fatto che i nodi correlati sono collegati fisicamente nel database, quindi l'accesso a tali relazioni è immediato quanto l'accesso ai dati stessi. L'utilizzo di un database a grafo consente di ottenere migliori performance lavorando su grandi quantità di dati.

3.2.2 Vantaggi dell'utilizzo di Neo4j

Neo4j è un database a grafo, di cui è possibile visualizzare la struttura tramite l'immagine 3.4, creato appositamente per gestire grandi quantità di dati altamente connessi che apporta grandi vantaggi:

- **Modello dei dati flessibile**.Neo4j fornisce un modello dati flessibile, semplice ma potente, che può essere facilmente modificato in base alle applicazioni ed ai settori.

- **Approfondimento in tempo reale**.Questo database è altamente disponibile per applicazioni in tempo reale di grandi aziende con garanzie transazionali.

- **Alta disponibilità**.Neo4j fornisce risultati basati su dati in tempo reale.

- **Dati connessi e semi-strutturati**.utilizzando Neo4j si possono rappresentare facilmente dati connessi e semi-strutturati.

- **Recupero facile**.Oltre a rappresentare i dati connessi, Neo4j permette di recuperare facilmente i dati collegati più velocemente rispetto ad altri database.

- **Linguaggio di interrogazione cifrato**. Neo4j fornisce un linguaggio di interrogazione dichiarativo per rappresentare visivamente il grafico, usando una sintassi ascii-art. I comandi di questo linguaggio sono in formato leggibile dall'uomo e molto facili da imparare.

- **Nessun join**. Consente un facile recupero di dati connessi/correlati.

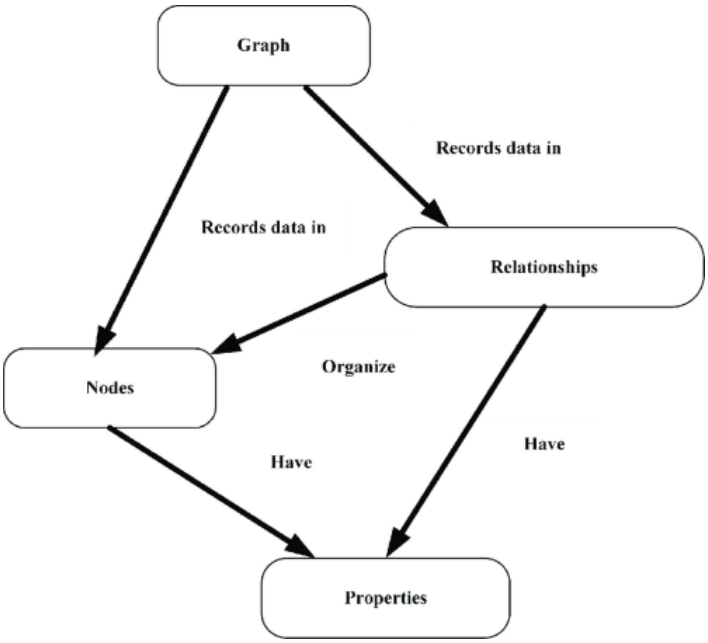

Figure 3.4: Schematizzazione delle memorizzazione e rappresentazione di dati di Neo4j.

3.3 Cypher

Cypher è un linguaggio di query dichiarativo specificamente progettato per gestire l'interrogazione dei database a grafo in modo efficiente. La sua creazione è ispirata da SQL, quindi consente di concentrarsi su quali dati si desiderano ottenere dal grafo e non su come ottenerli. Ciò consente di concentrarsi sul dominio del problema invece di preoccuparsi della sintassi. Cypher è stato progettato per essere facile da imparare ma molto potente quando si tratta di analisi dei grafi. La sua facilità è dovuta alla somiglianza con altri linguaggi e quindi alla sua intuitività. Ciò significa che Cypher può essere utilizzato per scrivere query complesse in modo relativamente semplice per database a grafo. Nella maggior parte dei casi, quando si lavora su testo, i nomi rappresentano i nodi del grafico, i verbi le relazioni nel grafico e gli aggettivi e gli avverbi le proprietà. Come ben noto, un database a grafo è composto da nodi, le quali etichette ne definiscono il tipo, e dalle relazioni tra essi. Sia che per etichette, proprietà che relazioni, Cypher è case sensitive, ovvero tratta maiuscole e minuscole nello stesso modo. Cypher, permette di effettuare interrogazioni su database a grafo utilizzando un'istruzione che generalmente si compone di due parti, una di interrogazione ed un'istruzione di ritorno, utilizzata per indicare eventuali valori da restituire. In ogni query opzionalmente si possono inserire uno o più comandi utilizzando le proprietà che si interessano ricercare.

Ovviamente, più istruzioni possono essere concatenate e si possono utilizzare valori precedentemente letti dal database. Cypher utilizza un tipo di sintassi ASCII-art per rappresentare i modelli dei grafi come segue:

```
{(n)-[:CONNESSO_A]->(m)}
```

Dove `n` ed `m` sono due nodi connessi tra loro. Le nozioni fondamentali da ricordare per un'istruzione Cypher sono che i nodi sono rappresentati da parentesi, le relazioni sono rappresentate da frecce e le informazioni su una relazione possono essere inserite tra parentesi quadre. Un esempio di query Cypher con la relativa rappresentazione grafica di nodi e relazioni è fornita dall'immagine 3.5.

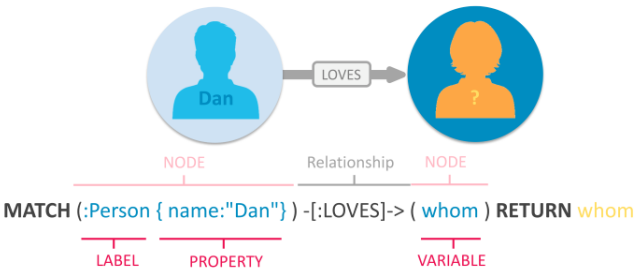

Figure 3.5: Rappresentazione grafica di nodi e relazioni con relativo esempio di query Cypher.

Sostanzialmente gli utenti di Neo4j utilizzano Cypher per costruire query espressive ed efficienti per eseguire qualsiasi tipo di creazione, lettura, aggiornamento o eliminazione (CRUD) sul loro grafo e Cypher è l'interfaccia principale per Neo4j. L'adozione di Cypher come linguaggio di interrogazione consente di rispondere a domande complesse velocemente, accelererando i tempi di elaborazione.Inoltre utilizzando Cypher, che è semplice da leggere e da scrivere, assieme a Neo4j che è senza schema, si ottiene la capacità di riprogettare completamente il modo in cui i dati sono collegati dopo aver caricato tutti i dati. Questi vantaggi contano molto durante le fasi di progettazione e di manutenzione di un'applicazione o di uno script.

3.4 Docker

Docker è una piattaforma open source per lo sviluppo, la creazione e l'esecuzione di applicazioni con la massima rapidità. Docker offre la possibilità di impacchettare ed eseguire un'applicazione in unità standardizzate utilizzando un'architettura client-server, quest'ultima può essere osservata nell'immagine

3.6. Il cuore del sistema Docker è rappresentato da un **Docker Engine**, anche chiamato **DE**, installato sulla macchina host e costituito da tre componenti:

- **Server**.Ovvero il demone Docker (dockerd), responsabile della creazione e della gestione dei contenitori.
- **Rest API**.Si occupa di stabilire la comunicazione tra i programmi e Docker e indica a dockerd cosa fare.
- **Command Line Interface**.Utilizzata per eseguire i comandi Docker.

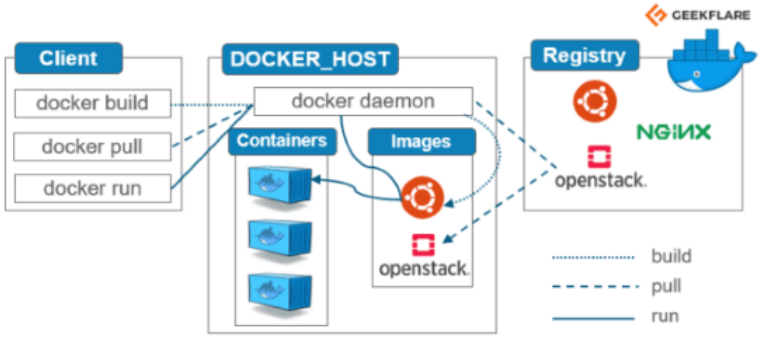

Figure 3.6: Architettura client-server utilizzata da Docker.Il client Docker comunica con il daemon Docker, che si occupa del lavoro pesante di creazione, esecuzione e distribuzione dei container Docker.

Le unità fondamentali di esecuzione necessarie per il funzionamento di Docker sono le **Docker images** e i **Docker container**. Una Docker image è un modello di sola lettura contenente le istruzioni per creare un Docker container. Un Docker container è un'istanza eseguibile di una Docker image che include tutto il necessario per l'esecuzione di un'applicazione, comprese le librerie, gli strumenti di sistema, ed il codice. I Docker container sono dei processi leggeri, isolati, effimeri, che non consentono persistenza dei dati nel tempo. Se, ad esempio, si vogliono rendere persistenti dati e moduli per un determinato script python risulta necessario creare una Docker image a partire da un immagine esistente arricchita con informazioni aggiuntive. Per creare un'immagine docker il primo passo è creare un Dockerfile, ovvero un normale file di testo contenente una serie di istruzioni, che vengono eseguite in successione per creare un'immagine Docker. Queste istruzioni includono la specifica del sistema operativo, delle lingue, delle variabili di ambiente Docker, dei percorsi dei file, delle porte di rete e di altri componenti necessari per eseguire l'immagine. Un Dockerfile specifica due cose per la creazione della nuova immagine:

- l'immagine di base da cui deriva la nuova (utilizzando il comando FROM).
- Una serie di modifiche specifiche che distinguono la nuova immagine da quella di base.

3.4.1 Docker e Neo4j

Come spiegato in precedenza, dentro la cartella `/mnt/ssd/barberini` è stato scaricato CKG. A questo punto utilizzando Docker possiamo iniziare a lavorare sul dataset caricandolo sul server neo4j lanciando il seguente comando:

```
docker run -it -p 37336:7687 -p 37339:7474
-d -v /mnt/ssd/barberini/data:/data
-v /mnt/ssd/barberini/dataset/:/import neo4j:3.5.18
```

Prima di procedere è necessario fornire una spiegazione dei flag utilizzati:

- `-it` : indica al Docker che dovrebbe aprire un'istanza del Docker container iterattivo con cui potersi interfacciare;
- `-p` : specifica la porta del Docker container a cui esporsi;
- `-d` : scollega il Docker container per l'esecuzione in background;
- `-v` : argomento necessario per creare spazio di archiviazione all'interno di un contenitore separato dal resto del filesystem del contenitore.

Una volta eseguito il comando precedente è possibile collegarsi alla pagina `http://137.204.107.153:37339/browser/` e autentificarsi con nome e password per visualizzare nodi, etichette e relazioni del dataset CKG. Una volta effettuato il passaggio di autenticazione la pagina si presenta come l'immagine 3.7.

La pagina presenta diverse icone, ad esempio cliccando sull'icona in alto a sinistra si può visualizzare l'elenco di tutte le etichette, anche chiamata labels, e delle relative relazioni presenti all'interno del grafo. Inoltre è presente una linea di codice nella quale è possibile eseguire query Cypher per effettuare operazioni sui vari nodi. Le principali query di interesse sono:

- `MATCH(n) RETURN n LIMIT X`: che permette di stampare un numero X di nodi e di visualizzarne i rispettivi attributi.
- `MATCH(n)-[r]->(m) RETURN n.name, type(r), m.name LIMIT X`: consente di visualizzare X triplette casuali, dove ogni tripletta è costituita da due nodi, ovvero n ed m, ed il tipo di relazione che li lega, ovvero type(r).

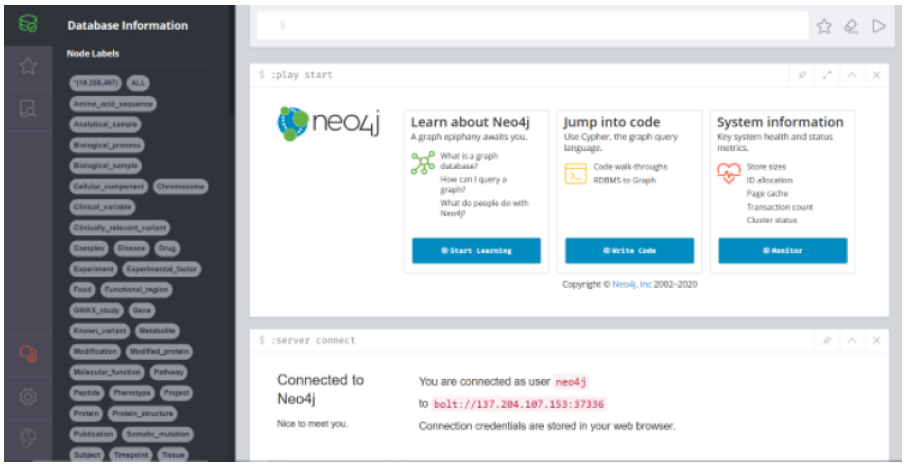

Figure 3.7: Pagina web Neo4j aperta dopo l'autenticazione.

- `MATCH(nid/name: '...')-[r]->(m) RETURN n, type(r), m`: visualizza tutti i nodi m ed il tipo di relazione che li collega ad una specifica entità n, identificata tramite id o nome.

- `MATCH(nid/name: '...')-[r id/name: '...']->(m) RETURN n, type(r),m`: stampa i nodi m collegati ad n tramite uno specifico tipo di relazione identificata dal suo id o dal suo nome. L'entità n, sulla quale viene effettuata la ricerca, viene specificata utilizzando o il nome o il suo id.

- `MATCH(nid/name: '...')-[r]->(mid/name: '...') RETURN type(r)`: permette di verificare se tra due entità vi sono relazioni esistenti.

- `MATCH p=(n)-[r]->(m) RETURN p LIMIT X`: utilizzata per estratte un subgrafo con profondità X a partire dal grafo originario.

3.5 Python3

Python3 è una nuova versione del linguaggio Python incompatibile con la versione 2. Il linguaggio rimane per lo più lo stesso, ma molti dettagli, in particolare il funzionamento di oggetti integrati, come dizionari e stringhe, sono stati modificati considerevolmente e molte funzionalità obsolete sono state finalmente rimosse. Python3 è stato scelto come linguaggio di programmazione da utilizzare in questo progetto perché è:

- **Interpretato**, ovvero non è necessario compilare il programma prima di eseguirlo.

- **Interattivo**, è permesso di interagire direttamente con l'interprete per scrivere i propri programmi.
- **Orientato agli oggetti**, poiché supporta lo stile o la tecnica di programmazione orientata agli oggetti che incapsula il codice all'interno degli oggetti.
- **Un linguaggio per principianti**, dato che supporta lo sviluppo di un'ampia gamma di applicazioni, a partire dalla semplice elaborazione del testo ai giochi.

Questo tipo di linguaggio, inoltre, venne progettato per essere altamente leggibile, grazie al frequente utilizzo di parole chiavi inglese e alla minor presenza di costruzioni sintattiche rispetto ad altre lingue.

3.5.1 Pycharm

Pycharm è l'IDE (Integrated Development Enviroment) selezionato per la scrittura di codice in linguaggio python. Pycham contiene sia un editor che un debugger molto completo ideato da JetBrains,per giunta è stato progettato da programmatori per programmatori così da poter fornire tutti gli strumenti necessari per uno sviluppo produttivo di Python. L'IDE in questione supporta le principali piattaforme di condivisione e revisione del codice, tra cui GitHub, e fornisce diverse funzioni per il miglioramento dell'efficienza, tra cui completamento intelligente del codice, correzioni rapide di errori, ispezione e refactoring del codice

3.5.2 Py2neo

Py2Neo è una libreria client e un toolkit che consente di lavorare con Neo4j all'interno delle applicazioni Python. Può essere installata facilmente utilizzando il seguente comando:

```
pip install --upgrade py2neo
```

La libreria supporta sia il protocollo Bolt che il protocollo http e fornisce un'API di alto livello, strumenti di amministrazione ed un supporto Cypher. Py2Neo è costituito da numerose API. Il cuore ed elemento fondamentale della libreria è l'API Graph che rappresenta un'istanza del database a grafo Neo4j. La sintassi per la creazione del grafo richiede di passare al costruttore sia l'URI, necessario per stabilire la connessione al grafo Neo4j, sia le credenziali per l'autentificazione, ovvero username e password:

```
graph = Graph(URI, auth=(username, password))
```

I due elementi costituitivi essenziali del modello a grafo etichettato sono il nodo e la relazione, due oggetti di py2neo contenuti nel pacchetto **data** e rispettivamente denominati **Node** e **Relationship**. Il nodo è l'unità primaria di memorizzazione dei dati all'interno di un grafo, può contenere un insieme di proprietà e può avere una o più etichette testuali. Una relazione è una connessione tipizzata e diretta tra una coppia di nodi, che può contenere una o più proprietà. Per poter interrogare un grafo su nodi e relazioni viene utilizzato Cypher. Prima di tutto è necessario scrivere la query per poi passarla al grafo ed eseguirla nel seguente modo:

```
graph.query(query)
```

All'interno del progetto viene utilizzato py2neo per poter creare script Python nei quali eseguire query Cypher su un database a grafo Neo4j.

3.6 SpaCy

SpaCy è una libreria open source gratuita per attività di elaborazione del linguaggio naturale in python. SpaCy è progettato specificatamente per l'utilizzo in fase di produzione e risulta essere un 'aiuto per lo sviluppo di applicazioni che processano e devono capire grandi volumi di testo. Può essere utilizzato per creare sistemi di estrazione delle informazioni o di comprensione del linguaggio naturale o per pre-elaborare il testo per il deep learning. Chiaramente spaCy non è un'API, non è una compagnia ma soprattutto non è un software di ricerca. SpaCy può essere di supporto per diverse attività di NLP come Tokenization, POS, Dependency Parsing, Lemmatization, NER, Text Classification, ecc.... Alcune delle operazioni offerte da spaCy funzionano in modo indipendente, mentre altre richiedono il caricamento di pipeline addestrate che consentono , ad esempio, a spaCy di prevedere annotazioni linguistiche (capire se una parola è un verbo o un sostativo). SpaCy è in grado di riconoscere vari tipi di entità in un documento chiedendo al modello di fare una predizione. Poiché i modelli utilizzati sono statistici e fortemente dipendenti dagli esempi sui quali sono allenati, questa operazione ha un'alta qualità e performance molto alte. Per l'elaborazione di testi biomedici o clinici le migliori performance si ottengono con **scispaCy**[31], ovvero un pacchetto Python contenente modelli pre-allenati specifici per l'ambito scientifico. All'interno del progetto è stato utilizzato scispaCy per l'attività di NER, dove una named entity è un'entità è un oggetto del mondo reale al quale è assegnato un nome, per esempio una persona, un animale, una nazione.

3.7 Applicazioni

In questa fase l'obbiettivo è quello di poter interrogare il grafo CKG, riprodotto su Neo4j, utilizzando query Cypher. Innanzitutto è necessario creare un'immagine Docker per lo sviluppo di Neo4j, a tal fine deve essere creato un file di testo, ovvero un Dockerfile specifico contenente sia il modulo di python3 che quello di py2neo: immagine Docker file A partire dal Dockerfile creato in precedenza è stata costruita l'immagine Docker, come mostrato:

```
docker build. -t neo4j
```

Il comando precedente è utilizzato per creare una Docker image a partire da un DockerFile contenuto nella directory locale. L'impiego del flag `-t` permette di specificare un nome, in questo caso neo4j, e di associarlo alla Docker image che si vuole creare. A questo punto per eseguire l'immagine all'interno di un contenitore viene lanciato il seguente comando:

```
docker run -rm -it -v $PWD:/workspace neo4j bash
```

I significati dei flag utilizzati sono già stati spiegati nella sezione 3.4.1, ad eccezione di `-rm` che dice al Daemon Docker di ripulire il contenitore e rimuovere il file system una volta che il contenitore viene chiuso. A questo punto il sistema è pronto per la creazione degli script python necessari per effettuare lo studio dei nodi, delle relazioni e delle etichette del grafo CKG. All'interno di quest'ultimo, come detto in precedenza, ogni nodo può avere un'etichetta, anche chiamata label, scelta tra le seguenti:
-ALL
-Amino_acid_sequence
-Analytical_sample
-Biological_process
-Biological_sample
-Cellular_component
-Chromosome
-Clinical_variable
-Clinically_relevant_variant
-Complex
-Disease
-Drug
-Experiment
-Experimental_factor
-Food
-Functional_region
-GWAS_Study

-Gene
-Known_variant
-Metabolite
-Modification
-Modified_protein
-Molecular_function
-Pathway
-Peptide
-Phenotype
-Project
-Protein
-Protein_structure
-Publication
-Somatic_mutation
-Subject
-Timepoint
-Tissue
-Transcript
-User

Per velocizzare l'esecuzione delle query è stato necessario creare un'etichetta univoca per tutti i nodi all'interno del grafo ed un indice customizzato. Come etichetta univoca si è scelto di utilizzare la label `ALL`, già presente all'interno del grafo ma inutilizzata. Prima di procedere con lo spostamento dei nodi si è notato che quelli etichettati come Known-variant o publication non contenevano informazioni utili, motivo per cui si è deciso di ignorare queste due etichette. Known-variant conteneva principalmente codici genetici senza descrizione, mentre publication era costituito da paper con solo le meta. I restanti nodi sono stati spostati sotto la label ALL, utilizzando il seguente script python:

```
from py2neo import Graph

graph = Graph("bolt://137.204.107.153:37336",
        auth=("neo4j", "generic"))
for l in ['Disease', 'Tissue', 'Pathway',
'Biological_process', 'Project', 'Transcript',
'Experiment', 'Experimental_factor', 'Protein',
'Functional_region', 'Metabolite', 'Phenotype',
'GWAS_study', 'Molecular_function', 'desease',
'Chromosome', 'Modified_protein', 'Chromosome',
'Modification', 'Cellular_component', 'User',
'Biological_sample', 'Somatic_mutation', 'ALL',
'Clinical_variable', 'Complex', 'Timepoint'
```

```
'Protein_structure', 'Amino_acid_sequence',
'Clinically_relevant_variant', 'Food','Node'
'Analytical_sample', 'Subject', 'Peptide',
'Gene', 'Drug', ]:
    query = f'MATCH(n:l) set n:l:ALL'
    graph.run(query)
```

Altro passaggio fondamentale è stata la creazione di un indice customizzato sulla proprietà `name` di ciascun nodo. Utilizzando gli accorgimenti appena elencati i tempi di elaborazione delle query sono stati ridotti al minimo consentendo una maggiore efficienza. Un'altra operazione molto importante è stata la creazione della relazione "is synonymous of". Questa relazione è creata tra due nodi quando uno è sinonimo dell'altro come può accadere per esempio con cancro e tumore.

Capitolo 4

Metodi ed esperimenti

Il metodo utilizzato mira a integrare una query con termini biomedici contenuti in un grafo di conoscenza esterno. A tal fine il primo passaggio consiste nell'estrazione delle entità fondamentali dalla query di partenza svolgendo l'attività di NER. Ciascuna delle entità selezionate comporta la creazione di un sottografo tramite l'utilizzo di un KG. I sottografi dovranno essere sottoposti ad un processo di linearizzazione per permettere l'effettuazione di esperimenti utilizzando i Language Model. Gli esperimenti permettono di trovare il valore ottimale degli iperparametri. L'obbiettivo finale consiste nel verificare se il nuovo input arricchito con informazioni aggiuntive comporti veramente un miglioramento delle performance.

4.0.1 Arricchimento del testo

In questa fase viene svolta l'attività di NER utilizzando un modello scispaCy, per poi effettuare le ricerche del sottografo sulle entità estratte tramite neo4j. L'algoritmo si compone di diversi passi, per prima cosa, per poter fare NER sul testo di partenza, si effettua il download del modello scispacy prescelto, ovvero en_core_sci_sm, una pipeline completa per l'elaborazione dei testi biomedici:

```
pip install https://s3-us-west-2.amazonaws.com/ai2-s2-
scispacy/releases/v0.5.1/en_core_sci_sm-0.5.1.tar.gz
```

Una volta terminato il download la pipeline viene caricata restituendo un oggetto, chiamato per semplicità *nlp*, che è un **language object** contenente tutte le componenti necessarie per l'elaborazione del testo, nel seguente caso per l'estrazione delle entità. Chiamando l'oggetto nlp sul testo di partenza si ottiene un documento elaborato, denominato *doc*. In seguito accedendo alla proprietà *ents* di *doc* vengono estratte le entità riconosciute dal language object. A questo punto tramite la classe Graph di py2neo si instaura la connessione

a neo4j, il database a grafo sul quale è riprodotto CKG. Per ciascuna delle entità estratte precedentemente, tramite l'attività di NER, si individuano tre entità collegate ad esse tramite uno specifico tipo di relazione. A tal scopo è necessaria la costruzione di una query cypher:

```
MATCH(n:ALL{name:entity_name})-[r]->(m)
RETURN n, type(r),m LIMIT 3
```

Il risultato ottenuto dall'esecuzione della query è un sottografo di dimensione pari a tre, dove le entità sono rappresentate da nodi interconnessi tra di loro. L'algoritmo appena descritto è così implementato:

```
import spacy
import scispacy
import en_core_sci_sm
from py2neo import Graph
from py2neo.data import Node, Relationship

graph = Graph("bolt://137.204.107.153:37336",
        auth=("neo4j", "generic"))
nlp = spacy.load("en_core_sci_sm")

question = "Hypersensitivity reactions to  melphalan allergy
during treatment of multiple myeloma, cancer and leukemia "

doc = nlp(question)
for entity in doc.ents:
    el = entity.text
    query = "MATCH(n:ALLname:entity_name)-[r]->(m)
            RETURN n, type(r),m LIMIT 3 "
    result = graph.query(query)
    print(result)
```

Quindi prendendo in esempio la seguente frase *Hypersensitivity reactions to melphalan allergy during treatment of multiple myeloma, cancer and leukemia* , d'apprima viene eseguita l'attivita di NER consentendo l'estrazione delle seguenti entità: *Hypersensitivity reactions*, *melphalan allergy*, *treatment*, *multiple myeloma*, *cancer* e *leukemia*. Successivamente per ciascuna delle entità elencate avviene la creazione di un sottografo, ad esempio per "leukemia", il sottografo creato è mostrato nell'immagine 4.1.

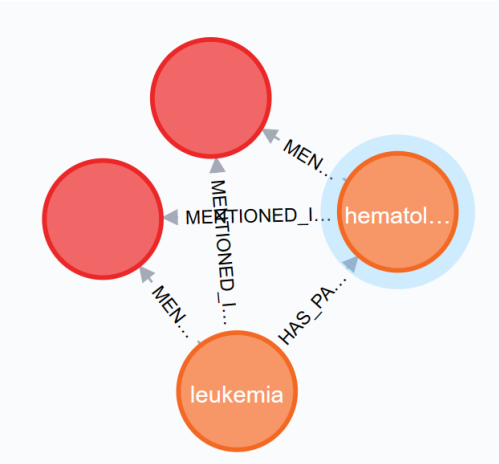

Figure 4.1: Sottografo relativo all'entità leukemia ottenuto utilizzando neo4j e CKG.

4.0.2 Linearizzazione

L'obbiettivo della linearizzazione è quello di trasformare i sottografi ottenuti in precedenza in una sequenza testuale che possa essere direttamente elaborata da un language model, come ad esempio BERT. La linearizzazione è compiuta attraversando ogni sottografo che rappresenta una delle entità individuate nella query di partenza. L'attraversamento consiste nell'individuare un nodo di partenza e i nodi collegati ad esso con le relative relazioni. Per ogni nodo di partenza sono stati selezionati sia il nome che l'etichetta, ovvero la categoria di appartenenza, che la descrizione. Le tre componenti sono state poi collegate secondo il seguente schema:

```
[NOME NODO] is considered [NODO LABEL]
and is described as [NODO DESCRIZIONE]
```

Successivamente per ognuno dei tre nodi collegati all'entità di partenza all'interno del sottografo oggetto di studio vengono estratti i tipi delle relazioni e i nomi dei nodi stessi. Questi elementi sono stati congiunti come segue:

```
[NOME NODO] has relation with [NOMI NODI VICINI]
```

Così facendo si ottiene un rappresentazione testuale che può essere direttamente elaborata dai language model. Il risultato prodotto dalla linearizzazione sulla frase presa di esempio per mostrare il funzionamento dell'algoritmo per l'arricchimento di testo è:

Hypersensitivity reactions to melphalan allergy during treatment of multiple myeloma, cancer and leukemia. melphalan allery is considered Disease and is described as a drug allergy that has allergic trigger melphalan. melphalan allergy MENTIONED_IN_PUBLICATION 29120401. melphalan allergy HAS_PARENT drug allergy. melphalan allergy is synonymous of melphalan allergy. treatment is considered Experimental_factor and is described as a process in which the act is intended to modify or alter some other material entity. treatment HAS_PARENT experimental process. treatment is synonymous of therapy. multiple myeloma is considered Disease and is described as a myeloid neoplasm that is located in the plasma cells in bone marrow . multiple myeloma HAS_PARENT myeloid neoplasm. multiple myeloma HAS_PARENT leukocyte disease. multiple myeloma MENTIONED_IN_PUBLICATION 2208106. cancer is considered Disease and is described as a disease of cellular proliferation that is malignant and primary, characterized by uncontrolled cellular proliferation, local cell invasion and metastasis. cancer HAS_PARENT disease of cellular proliferation. cancer HAS_PARENT neoplasm. cancer MAPS_TO organ system cancer. leukemia is considered Disease and is described as a cancer that affects the blood or bone marrow characterized by an abnormal proliferation of blood cells. leukemia HAS_PARENT hematologic cancer. leukemia MENTIONED_IN_PUBLICATION 1066182. leukemia MENTIONED_IN_PUBLICATION 100897.

4.1 Esperimenti

4.1.1 Setup

BioASQ

Il dataset scelto per il progetto è BioASQ[32] che valuta la capacità dei sistemi di indicizzare semanticamente un numero molto elevato di articoli scientifici biomedici e di restituire risposte concise e comprensibili all'utente a determinate domande in linguaggio naturale combinando informazioni da articoli biomedici e ontologie. In particolare è stato seleionato il task 8 composto da:

- **TaskA**. Attività che ha l'obbiettivo di favorire l'indicizzazione di grandi volumi di dati non etichettati, principalmente articoli scientifici, con concetti biomedici. Comprende un set di dati di circa 14,9 milioni di articoli annotati.

- **TaskB**. Questa attività utilizza set di dati di benchmark contenenti domande in ambito biomedico con le rispettive risposte. Permette l'elaborazione di domande biomediche e la generazione di risposte.

Per effettuare il download del dataset è necessario seguire 4 passaggi:

- **1** . Registrarsi al link:
  http://participants-area.bioasq.org/datasets/%5D.
- **2**. Scaricare il file di training, ovvero *traning8b.json*.
- **3**. Scaricare i dati del test, ovvero il *task8a.json*.
- **4**. Impostare ed eseguire lo script di preelaborazione:

  ```
  python3 preprocessing.py
  ```

  Il file preprocessing.py è utilizzato per estrarre tutti gli articoli pubmed per il task8B, in particolare per ogni articolo si occupa di selezionare pmid, titolo ed abstract.

Per ognuno dei file scaricati è stata studiata la struttura utilizzando un file python, ovvero *info.py*. Per il file relativo al taskA si è analizzato il numero totale di articoli e la loro lunghezza media, per i file relativi al task8B, invece, si è studiato il numero totale di domande, la loro lunghezza media, il numero medio di possibili risposte e la loro lunghezza media.

Triplet Loss

La funzione di loss viene utilizzata nella fase di fine-tuning per addestrare i modelli prescelti a produrre incorporamenti più vicini all'ancora per il caso positivo rispetto al caso negativo, per il progetto è stato scelto di utilizzare la Triplet Loss. L'implementazione di quest'utlima richiede l'utilizzo di due moduli di pytorch, ovvero losses e miners. Le funzioni di mining accettano un batch di n incorporamenti e ritornano k coppie/triplette da utilizzare per il calcolo della funzione di loss. Utilizzando come funzione di loss la TripletMarginLoss la funzione di mining scelta, ovverp TripletMarginMiner, genera k triplette.

```
miner = miners.TripletMarginMiner(margin=0.2,
                        type_of_triplets="all")
loss_func = losses.TripletMarginLoss(triplets_per_anchor=10)
```

Quindi, utilizzando miner, loss, embeddings e rispettive etichette la funzione di loss è stata implementata come segue:

```
loss = loss_func(embeddings, labels, mined)
```

Impostazione del progetto

Per poter scaricare il progetto è necessario clonare la repository github sul proprio dispositivo utilizzando il comando:

```
git clone https://github.com/valgi0/KB-injection-bio-dpr
```

Successivamente ci si deve spostare all'interno della cartella clonata KB-injection-bio-dpr e costruire l'immagine Docker come segue:

- **1**. Creare una cartella chiamata *build*:

  ```
  mkdir build
  ```

- **2**. Posizionare il file *.netrc* nella directory build per poter accedere a wandb.

- **3**. Aprire il file .netrc e impostare le credenziali per l'autentificazione, ovvero username e password.

- **4**. Copiare il dockerfile nella cartella build:

  ```
  cp Dockerfile ./build
  ```

- **5**. Costruire l'immagine docker:

  ```
  cd ./build; docker build . -t kb-bio-dpr.
  ```

- **6**. Controllare che l'immagine Docker sia stata creata correttamente:

  ```
  docker image ls
  ```

A questo punto può essere avviato il container docker dove testare i modelli:

```
docker run --rm -it --gpus all -v $PWD:/workspace -v
DATA_PATH:/data kb-bio-dpr bash
```

Ad esempio utilizzando BioBERT l'allenamento della DPR viene effettuato usando il seguente comando all'interno del docker container:

```
python3 main.py --path_to_pubmed_json
/data/taska_articles.json --path_to_task_b_json
/data/training8b.json --gpu 1 --no_wandb --run_name DPR-
BioBert --run_project KbBioDpr-train_dpr --
check_val_every_n_steps 300 --precision 16
```

4.1.2 Baseline

Transformers fornisce API e strumenti per scaricare e addestrare facilmente modelli preaddestrati all'avanguardia. L'utilizzo di modelli preaddestrati permette di ridurre i costi di elaborazione, di risparmiare il tempo e le risorse necessarie per addestrare un modello da zero. L'utilizzo dei modelli forniti da Transformers è possibile grazie ai moduli AutoTokenizer e AutoModel. I modelli selezionati dalla piattaforma HuggingFace da utilizzare all'interno del progetto sono:

- *emilyalsentzer/Bio_ ClinicalBERT*. Modello costruito sulle basi di BioBERT ed allenato su MIMIC, un database contenente cartelle cliniche elettroniche dei pazienti in terapia intensiva presso il Beth Israel Hospital di Boston.

- *allenai/scibert_ scivocab_ uncased*. Un modello BERT allenato su testo scientifico preso da Semantic scholar, ovvero uno strumento di ricerca gratuito basato sull'intelligenza artificiale per la letteratura scientifica.

- *michiyasunaga/LinkBERT-base*. LinkBERT è un miglioramento di BERT che acquisisce collegamenti ipertestuali e collegamenti di citazioni per includere la conoscenza estesa su più documenti. LinkBERT raggiunge rispetto a BERT prestazioni migliori per le attività di comprensione generale della lingua (ad es. classificazione del testo) ed è anche particolarmente efficace per attività ad alta intensità di conoscenza (ad es. risposta a domande) e attività tra documenti (ad es. comprensione della lettura, recupero di documenti).

- *microsoft/BiomedNLP-PubMedBERT-base-uncased-abstract-fulltext*. PubMedBERT è preaddestrato da zero utilizzando abstract estratti da PubMed e articoli full-text estratti da PubMedCentral. Questo modello raggiunge prestazioni all'avanguardia in molte attività di NLP biomediche e attualmente detiene il punteggio più alto nel Biomedical Language Understanding and Reasoning Benchmark.

- *cambridgeltl/SapBERT-from-PubMedBERT-fulltext*. SapBERT è addestrato su UMLS 2020AA utilizzando come modello base microsoft/BiomedNLP-PubMedBERT-base-uncased-abstract-fulltext.

I modelli sono stati scelti perché pre-allenati su dataset scientifici o biomedici e perché hanno recentemente raggiunto lo stato dell'arte in molti compiti di NLP in ambito biomedico.

4.1.3 Metriche di valutazione

Le metriche utilizzate per valutare le performance dei modelli sono le seguenti:

- **Map**. Mean average precision, è la metrica più popolare utilizzata dalle sfide di benchmark come ImageNET, Google Open Image Challenge, ecc. Un buon valore di Map indica che il modello è robusto e stabile. Per definire il termine, la precisione media (o mAP) è una metrica di Machine Learning progettata per valutare gli algoritmi di rilevamento degli oggetti che al giorno d'oggi viene utilizzata anche per valutare i modelli di istanza e segmentazione semantica.

$$Map = \frac{\sum_{q=1}^{Q} AveP(q)}{Q} \tag{4.1}$$

  dove Q è la lista di query mentre $AveP$ è la precisione media al punto q.

- **Recall**. Consiste nella capacità di un modello di trovare tutti i casi rilevanti all'interno di un set di dati. Un punteggio basso della recall indica che il modello di apprendimento automatico prevederà alcuni falsi negativi. I veri positivi sono punti dati classificati dal modello come positivi che in realtà sono positivi (nel senso che sono corretti) e i falsi negativi sono punti dati che il modello identifica come negativi che in realtà sono positivi (errati).Matematicamente, la recall può essere così definita:

$$Recall = TruePositives/(TruePositives + FalseNegatives) \tag{4.2}$$

- **Precision**. La precisione è un fattore essenziale da considerare quando si valutano le prestazioni di un modello di machine learning. È definito come la proporzione di veri positivi rispetto a tutte le previsioni positive, inclusi falsi positivi. I falsi positivi sono casi che il modello etichetta erroneamente come positivi ma che in realtà sono negativi. Una bassa precisione significa che il nostro modello di machine learning prevederà alcuni falsi positivi.

$$Precision = TruePositives/(TruePositives + FalsePositives) \tag{4.3}$$

- **Ndcg**. NDCG è l'acronimo di Normalized Discounted Cumulative Gain. Discounted Cumulative Gain (DCG) è la metrica per misurare la qualità del ranking. Viene utilizzato principalmente nei problemi di recupero delle informazioni come ad esempio per misurare l'efficacia dell'algoritmo

di un motore di ricerca classificando gli articoli visualizzati in base alla loro rilevanza in termini di parola chiave di ricerca. Il NDCG è il DCG con un fattore di normalizzazione al denominatore:

$$NDCG_K = \frac{\sum_{i=1}^{K} \frac{G_i}{\log_2(i+1)}}{\sum_{i=1}^{K} \frac{G_i^{ideal}}{\log_2(i+1)}} \tag{4.4}$$

dove K è il numero di elementi.

- **Bpref**. Bpref è una metrica che calcola una relazione di preferenza se gli elementi giudicati rilevanti vengono recuperati prima degli elementi giudicati irrilevanti. La misura bPref è definita come:

$$bpref = \frac{1}{R} \sum_{r} 1 - \frac{|nrankedhighetthenr|}{min(R, N)} \tag{4.5}$$

dove R è il numero di documenti giudicati rilevanti, N è il numero di documenti giudicati irrilevanti, r è un documento recuperato rilevante e n è un membro dei primi R documenti recuperati irrilevanti.

4.1.4 Tuning degli hyperparametri

Gli iperparametri fondamentali per il nostro lavoro sono *batch size*, denominato *bs*, e *learning rate*, denominato *lr*. Il learning rate è un iperparametro che controlla quanto modificare il modello in risposta all'errore stimato ogni volta che i pesi del modello vengono aggiornati. Il batch size definisce il numero di campioni su cui lavorare prima di aggiornare i parametri del modello interno. In parole povere, la dimensione del batch è il numero di campioni che verranno trasmessi alla rete contemporaneamente. La scelta del batch size è impegnativa in quanto un valore troppo piccolo può comportare un lungo processo di allenamento che potrebbe bloccarsi, mentre un valore troppo grande può comportare l'apprendimento troppo veloce di una serie di pesi non ottimale o un processo di allenamento instabile. Il tuning degli iperparametri è la fase in cui abbiamo testato le performance dei vari modelli selezionati variando i valori di lr e bs. Sostanzialmente abbiamo innanzitutto testato tutti i modelli selezionati, indicati nella sottosezione [**?**], con batch size pari a due e learning rate pari a 0.0001 e abbiamo selezionato i due con performance migliori, ovvero *microsoft/BiomedNLP-PubMedBERT-base-uncased-abstract-fulltext* e *allenai/-scibert_scivocab_uncased*. Ciascuno dei due modelli risultati più performanti è stato poi testato con bs fisso a 2 e diversi valori di learning rate utilizzando il seguente algoritmo:

```
for lr in 0.0005 0.0001 0.00005 0.00001 0.000005
0.000001 0.0000005 0.0000001 do
./run_on_docker.sh python3 main.py
--path_to_pubmed_json /data/taska_articles.json\
--path_to_task_b_json /data/training8b.json\
--path_to_task_b_json_test /data/testset/testset10000_linearization.json\
--path_to_pubmed_json_test /data/testset/pubmed_testset_10000.json\
--gpu 1
--run_name prova\
--run_project Kb-ablatives-lr-batch-try
--lr ${lr}
--batch_size 2\
--check_val_every_n_steps 300
--precision 16\
--model_name
--do_test \
--max_epochs 10\
--dont_save
done
```

Il flag `-run_name` indica il nome del file ed il nome dell'esecuzione su wandb mentre con il flag `-run_project` si specifica il nome della directory dove salvare il checkpoint su wandb. I risultati ottenuti sono poi stati confrontati consultando i grafici disponibili su wandb. I grafici in questione rappresentano le metriche citate nella sottosezione 4.1.3. L'analisi ha permesso di selezionare per ciascuno dei due modelli più performanti i due valori di learning rate migliori, ovvero 0.00001 e 0.000005 per BiomedNLP e 0.00001 e 0.00005 per sciBERT. Successivamente, sia per sciBERT che BiomedNLP, sono stati effettuati esperimenti utilizzando i valori di learning rate selezionati in precedenza associati a ciascuno dei valori di bs compreso tra 2 e 6. Così facendo, infine, si sono apprese le 2 combinazioni migliori di lr e bs per ciascun modello osservando i risultati ottenuti su wandb. Per SciBERT :

- **lr=0.00005 e bs=3**;
- **lr=0.00001 e bs=6**.

BiomedNLP ottiene invece le migliori performance nei seguenti casi:

- **lr=0.000005 e bs=6**;
- **lr=0.00001 e bs=3**.

Dai grafici in figura 4.2 si può osservare che il modello BiomedNLP ha ottenuto punteggi più alti rispetto a SciBERT. In particolare, il punteggio ottenuto nella recall indica, come spiegato nella sezione 4.1.3, che BiomedNLP ha una capacità superiore in ambito biomedico di estrarre documenti rilevanti. Questo concetto è confermato anche dal punteggio ottenuto da BiomedNLP rispetto alla metrica di ndcg.

4.1.5 Arricchimento del testo

L'obbiettivo finale del progetto è quello di verificare se utilizzando l'arricchimento del testo si ottengano performance migliori rispetto a quando non se ne fa uso utilizzando gli stessi modelli. A tal scopo le migliori combinazioni di lr e bs ottenute su BiomedNLP e sciBERT sono testate nuovamente utilizzando l'arricchimento di testo e la linearizzazione. Questo è possibile utilizzando lo stesso algoritmo ma con il flag `-use_linearization` che indica, appunto, che si deve utilizzare la versione linearizzata del dataset. Difatti per ciascun file del dataset BioASQ è stata creata una versione linearizzata utilizzando l'algoritmo spiegato nella sezione 4.0.2, sostanzialmente preso ad esempio il file *training8b.json* ne è stata creata una versione linearizzata chiamata *training8b_ linearization.json*. Assegniamo per facilità un denominativo a ciascun modello:

- **B1**, BiomedNLP con lr=0.00001 e bs=3.
- **B1L**, versione linearizzata di B1.
- **B2**, BiomedNLP con lr=0.000005 e bs=3.
- **B2L**, versione linearizzata di B2.
- **S1**, SciBERT con lr=0.00005 e bs=3.
- **S1L**, versione linearizzata di S1.
- **S2**, SciBERT con lr=0.00001 e bs=6.
- **S2L**, versione linearizzata di S2.

Osservando i grafici in figura 4.3 si può osservare che il modello BiomedNLP ha prestazioni migliori rispetto a SciBERT , questo è dovuto ai dataset con i quali i due modelli sono stati pre-addestrati. BiomedNLP è stato allenato su dataset biomedici, in particolare su articoli Pubmed, che risultano essere più adeguati per i dati biomedici, mentre per SciBERT sono stati utilizzati documenti scientifici. I risultati ottenuti da BiomedNLP con linearizzazione risultano

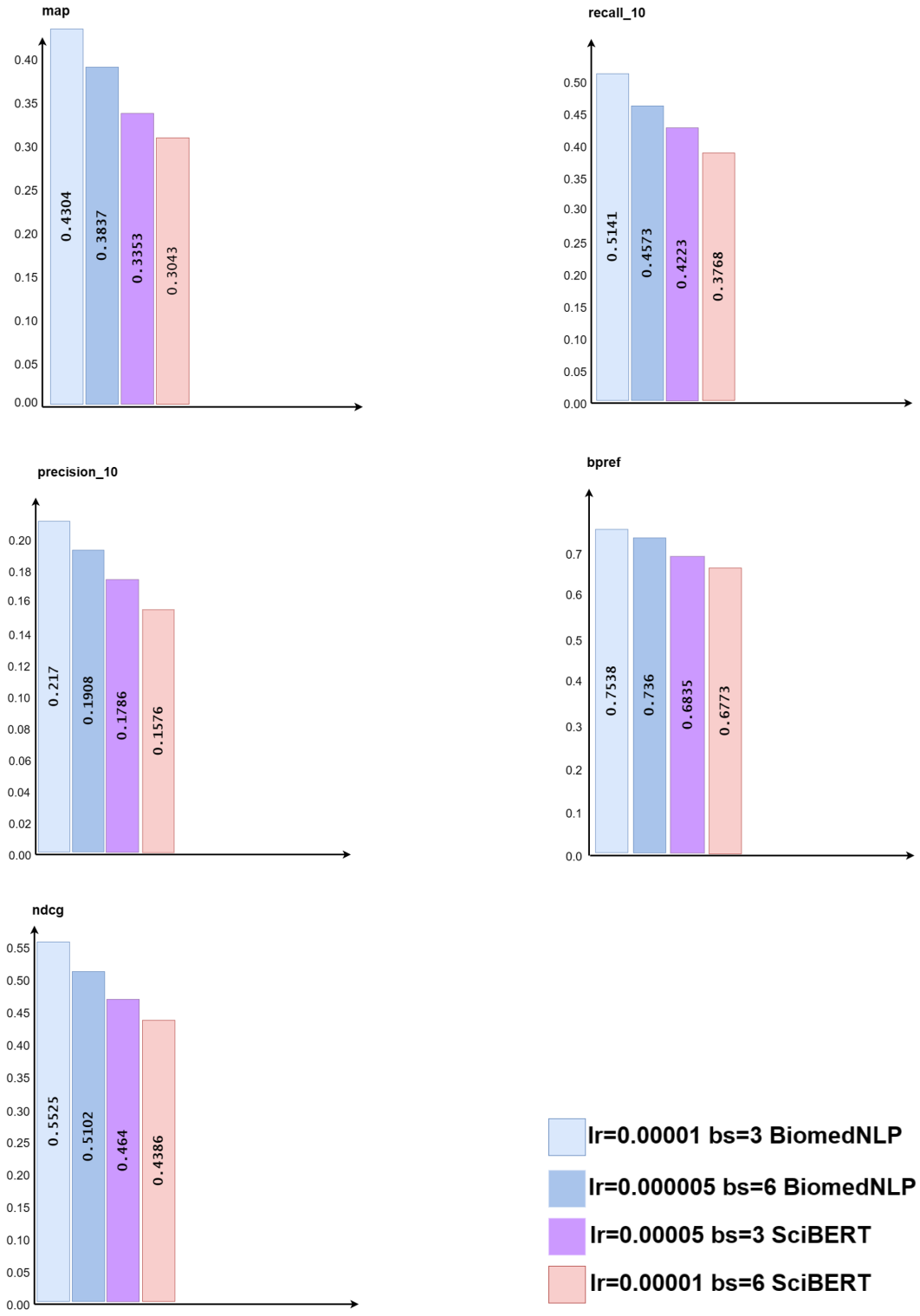

Figure 4.2: Grafici relativi alle metriche di valutazione per i modelli testati senza linearizzazione.

essere peggiori al contrario di SciBERT dove le performance sono mediamente migliori utilizzando la soluzione proposta. Concentrando l'attenzione su bpref, metrica che considera se i documenti rilevanti sono classificati al di sopra di quelli irrilevanti, il modello *B1L* ottiene però un punteggio superiore rispetto a *B1*. Il punteggio ottenuto da *B1L* nella bpref denota una sua maggiore capacità nel selezionare e prendere maggiormente in considerazione documenti rilevanti in ambito biomedico. Spostando l'attenzione su S2, si nota che la versione linearizzata, S2L, ottiene punteggi superiori nelle seguenti metriche: map, recall_10, precision_10. La maggior precisione delle previsioni, il maggior numero di previsioni corrette e la maggior stabilità di S2L permettono di definire le sue performance superiori rispetto a quelle di S2. La linearizzazione comporta un aumento di informazioni, ma solo in alcuni casi può essere utile per ottenere performance migliori da parte dei modelli, come per S2.

Alla luce dei risultati sopra esposti, 4.3, e considerando che SciBERT ottiene prestazioni mediamente migliori utilizzando la linearizzazione e che esso è un dataset non specifico per l'ambito biomedico, possiamo dire che la nostra soluzione può essere usata nella fase di fine-tuning per modelli non biomedici in ambito biomedico. In questo caso il modello estende le proprie conoscenze essendo così in grado di capire meglio il significato, basandosi anche sul contesto, di termini che non conosce all'interno di una frase. Al contratrio, è interessante notare che un modello preaddestrato sul dominio biomedico, e che quindi contiene già al suo interno le conoscenze specifiche, non beneficia di questo arricchimento.

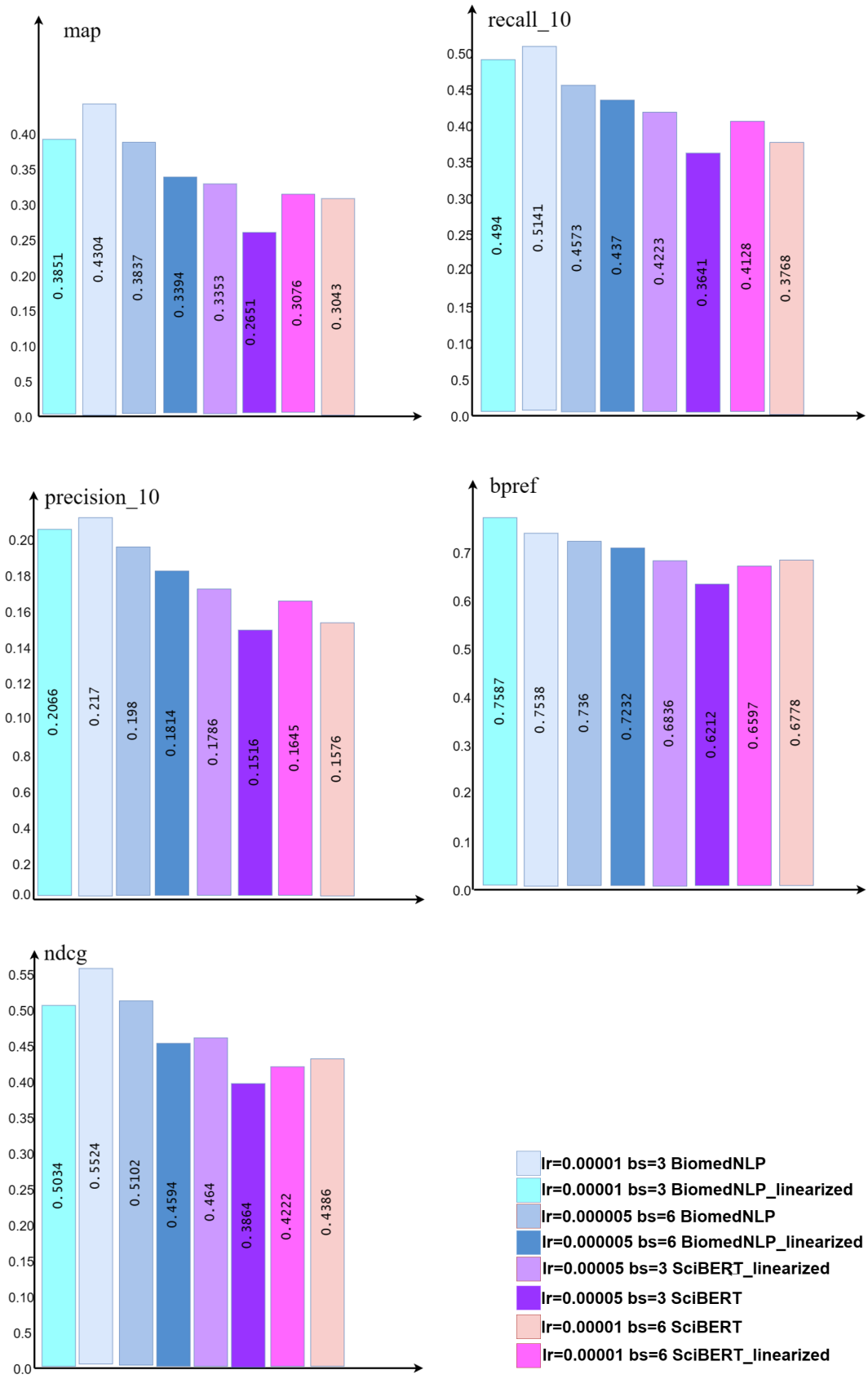

Figure 4.3: Grafici relativi alle metriche di valutazione per i modelli testati con e senza linearizzazione.

Capitolo 5

Conclusioni

Il progetto pone l'attenzione sull'effettuare studi nell'ambito del deep learning applicato al Natural Language Processing. L'obbiettivo è quello di verificare se utilizzando dei large language model (LLM), l'adozione di dataset arricchiti con informazioni su termini biomedici porta a performance più alte. A tal scopo è stato studiato ed implementato un algoritmo che presa una frase di input ne seleziona le entità rilevanti tramite l'attività di NER e per ognuna di esse crea un sottografo con informazioni aggiuntive prese da un knowledge graph esterno, nello specifico il CKG. I sottografi ottenuti subiscono poi un processo di linearizzazione prompt-based per produrre un risultato testuale che può essere direttamente utilizzato dai language model. Per questo progetto sono stati selezionati dei LLM pre-allenati su Hugginface che ad oggi rappresentano lo stato dell'arte, sia in campo biomedico che non. Abbiamo effettuato un primo esperimento con learning rate e batch size fisso per selezionare tra i precedenti i 2 migliori modelli, ovvero BiomedNLP e SciBERT. Questi ultimi sono stati sottoposti ad una fase di fine-tuning per ricercare i valori degli iperparametri migliori così da poter effettuare gli esperimenti anche sul dataset linearizzato e confrontare i risultati otetnuti. Alla luce dei risultati emersi BiomedNLP risulta essere il miglior modello, tra quelli testati, in ambito biomedico. Inoltre risulta che la nostra soluzione permetta di ottenere un miglioramento delle per database generici in ambito biomedico, come accade per SciBERT. In questo caso i modelli, utilizzando l'arricchimento di testo, sono in grado di acquisire nuove conoscenze su termini sconosciuti, migliorando le proprie conoscenze. Risultato interessante è quello di BiomedNLP che dimostra che modelli preaddestrati su domini biomedici, non beneficino del tutto di questo arricchimento. BiomedNLP è anche la dimostrazione che questo tipo di modelli può ottenere tramite l'arricchimento una capacità di scegliere documenti più precisi ed accurati, come dimostrato dal punteggio ottenuto con la metrica bpref nel grafico 4.3. Sicuramente questo lavoro ha sottolineato la necessità

di effettuare ulteriori ricerche per approfondire l'utilità dell'adozione di input arricchito in campo biomedico. Una delle possibilità più interessanti è quella di effettuare gli esperimenti con funzioni di loss differenti, come ad esempio la CrossBatchMemory o la VICReg, per poter studiare il comportamento degli stessi modelli già analizzati. Inoltre abbiamo potuto osservare che il modello che ha raggiunto i migliori risultati è stato BiomedNLP, soprattutto la versione con learning rate uguale a 0.00001 e batch size uguale a 3, ma sarebbe interessante effettuare i test anche con modelli che non rappresentano lo stato dell'arte sia pre-allenati con dataset di dominio biomedico che non.

Ringraziamenti

Ringrazio il professor Moro per la possibilità offerta con questa tesi di acquisire nuove competenze e conoscenze, per la disponibilità e l'assistenza e per aver alimentato il mio interesse nell'ambito della materia. Ringrazio Lorenzo, il mio correlatore, per la gentilezza ed il suo aiuto. Ringrazio la mia famiglia ed i miei amici per il supporto e per essermi sempre stati accanto.

Bibliografia

[1] Som Biswas. Chatgpt and the future of medical writing, 2023.

[2] Basemah Alshemali and Jugal Kalita. Improving the reliability of deep neural networks in nlp: A review. *Knowledge-Based Systems*, 191:105210, 2020.

[3] Biao Zhang, Barry Haddow, and Alexandra Birch. Prompting large language model for machine translation: A case study, 2023.

[4] Prakash M. Nadkarni, Lucila Ohno-Machado, and Wendy Webber Chapman. Natural language processing: an introduction. *J. Am. Medical Informatics Assoc.*, 18(5):544–551, 2011.

[5] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *Int. J. Mach. Learn. Cybern.*, 1(1-4):43–52, 2010.

[6] Shirui Wang, Wen'an Zhou, and Chao Jiang. A survey of word embeddings based on deep learning. *Computing*, 102(3):717–740, 2020.

[7] Abhishek Sharma, Sudeshna Chakraborty, and Shivam Kumar. Named entity recognition in natural language processing: A systematic review. In *Proceedings of Second Doctoral Symposium on Computational Intelligence: DoSCI 2021*, pages 817–828. Springer, 2022.

[8] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digit. Signal Process.*, 73:1–15, 2018.

[9] Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.

[10] Xiaohan Zou. A survey on application of knowledge graph. *Journal of Physics: Conference Series*, 1487(1):012016, mar 2020.

[11] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. Recent advances in natural language processing via large pre-trained language models: A survey. *CoRR*, abs/2111.01243, 2021.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[13] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. Multi-head attention: Collaborate instead of concatenate. *CoRR*, abs/2006.16362, 2020.

[14] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In Marilyn A. Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pages 464–468. Association for Computational Linguistics, 2018.

[15] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019.

[16] Sho Takase and Naoaki Okazaki. Positional encoding to control output sequence length. *CoRR*, abs/1904.07418, 2019.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[18] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on*

Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, 2019.

[19] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinform.*, 36(4):1234–1240, 2020.

[20] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3613–3618. Association for Computational Linguistics, 2019.

[21] Brian Kulis. Metric learning: A survey. *Found. Trends Mach. Learn.*, 5(4):287–364, 2013.

[22] Ivano Lauriola, Alberto Lavelli, and Fabio Aiolli. An introduction to deep learning in natural language processing: Models,techniques, and tools. *Neurocomputing*, 470:443–456, 2022.

[23] Mahmut Kaya and Hasan Sakir Bilge. Deep metric learning: A survey. *Symmetry*, 11(9):1066, 2019.

[24] Xun Wang, Xintong Han, Weilin Huang, Dengke Dong, and Matthew R. Scott. Multi-similarity loss with general pair weighting for deep metric learning. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 5022–5030. Computer Vision Foundation / IEEE, 2019.

[25] Amit Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[26] Bhaskar Mitra and Nick Craswell. An introduction to neural information retrieval. *Foundations and Trends® in Information Retrieval*, 13(1):1–126, December 2018.

[27] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online,*

November 16-20, 2020, pages 6769–6781. Association for Computational Linguistics, 2020.

[28] Samuel Humeau, Kurt Shuster, Marie-Anne Lachaux, and Jason Weston. Poly-encoders: Architectures and pre-training strategies for fast and accurate multi-sentence scoring. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[29] Alberto Santos, Ana R Colaço, Annelaura B Nielsen, Lili Niu, Maximilian Strauss, Philipp E Geyer, Fabian Coscia, Nicolai J Wewer Albrechtsen, Filip Mundt, Lars Juhl Jensen, et al. A knowledge graph to interpret clinical proteomics data. *Nature Biotechnology*, 40(5):692–702, 2022.

[30] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th annual Southeast regional conference*, pages 1–6, 2010.

[31] Mark Neumann, Daniel King, Iz Beltagy, and Waleed Ammar. Scispacy: Fast and robust models for biomedical natural language processing. In Dina Demner-Fushman, Kevin Bretonnel Cohen, Sophia Ananiadou, and Junichi Tsujii, editors, *Proceedings of the 18th BioNLP Workshop and Shared Task, BioNLP@ACL 2019, Florence, Italy, August 1, 2019*, pages 319–327. Association for Computational Linguistics, 2019.

[32] George Tsatsaronis, Georgios Balikas, Prodromos Malakasiotis, Ioannis Partalas, Matthias Zschunke, Michael Alvers, Dirk Weißenborn, Anastasia Krithara, Sergios Petridis, Dimitris Polychronopoulos, Yannis Almirantis, John Pavlopoulos, Nicolas Baskiotis, Patrick Gallinari, Thierry Artieres, Axel-Cyrille Ngonga Ngomo, Norman Heino, Eric Gaussier, Liliana Barrio-Alvers, and Georgios Paliouras. An overview of the bioasq large-scale biomedical semantic indexing and question answering competition. *BMC Bioinformatics*, 16:138, 04 2015.