ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria
Laurea Triennale in Ingegneria e Scienze Informatiche

SPERIMENTAZIONE DI DEEP METRIC LOSS PER SELF-SUPERVISED INFORMATION RETRIEVAL SYSTEMS SU CORD19

Elaborato in
Programmazione di Applicazioni Data Intensive

Relatore
Prof. Gianluca Moro

Co-relatore
Dott. Lorenzo Valgimigli

Presentato da
Enrico Brunetti

Seconda Sessione di Laurea
Anno Accademico 2020-2021

PAROLE CHIAVE

Natural Language Processing

Self-Supervised Learning

Deep Metric Learning

Transformers

IR

Sommario

Dopo lo sviluppo dei primi casi di Covid-19 in Cina nell'autunno del 2019, ad inizio 2020 l'intero pianeta è precipitato in una pandemia globale che ha stravolto le nostre vite con conseguenze che non si vivevano dall'influenza spagnola. La grandissima quantità di paper scientifici in continua pubblicazione sul coronavirus e virus ad esso affini ha portato alla creazione di un unico dataset dinamico chiamato CORD19 e distribuito gratuitamente. Poter reperire informazioni utili in questa mole di dati ha ulteriormente acceso i riflettori sugli information retrieval systems, capaci di recuperare in maniera rapida ed efficace informazioni preziose rispetto a una domanda dell'utente detta query. Di particolare rilievo è stata la TREC-COVID Challenge, competizione per lo sviluppo di un sistema di IR addestrato e testato sul dataset CORD19. Il problema principale è dato dal fatto che la grande mole di documenti è totalmente non etichettata e risulta dunque impossibile addestrare modelli di reti neurali direttamente su di essi. Per aggirare il problema abbiamo messo a punto nuove soluzioni self-supervised, a cui abbiamo applicato lo stato dell'arte del deep metric learning e dell'NLP. Il deep metric learning, che sta avendo un enorme successo soprattuto nella computer vision, addestra il modello ad "avvicinare" tra loro immagini simili e "allontanare" immagini differenti. Dato che sia le immagini che il testo vengono rappresentati attraverso vettori di numeri reali (embeddings) si possano utilizzare le stesse tecniche per "avvicinare" tra loro elementi testuali pertinenti (e.g. una query e un paragrafo) e "allontanare" elementi non pertinenti. Abbiamo dunque addestrato un modello SciBERT con varie loss, che ad oggi rappresentano lo stato dell'arte del deep metric learning, in maniera completamente self-supervised direttamente e unicamente sul dataset CORD19, valutandolo poi sul set formale TREC-COVID attraverso un sistema di IR e ottenendo risultati interessanti.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Negli anni '50 il matematico inglese Alan Turing, noto per aver decifrato il codice di comunicazione Enigma, utilizzato dalle potenze dell'Asse durante la Seconda Guerra Mondiale, propose l'idea di una macchina che apprende, ovvero in grado di imparare e diventare intelligente. Tuttavia, in quel periodo, questo tipo di studi si stava concentrando su approcci logici di tipo knowledge-based e si dovranno aspettare gli anni '90 per la fioritura vera e propria dell'apprendimento automatico, il **machine learning**, i.e. una branca dell'intelligenza artificiale che fornisce metodi generali per estrarre modelli di conoscenza dai dati (e.g., classificare recensioni in positive e negative avendo a disposizione il testo).

I tipi di apprendimento automatico vengono generalmente suddivisi in tre diverse categorie:

- **Supervised Learning**. Al modello vengono forniti degli esempi sotto forma di input e i rispettivi output desiderati, mentre l'obiettivo è quello di estrarre una regola generale che associ l'input all'output corretto.
- **Unsupervised Learning**. Il modello ha lo scopo di trovare una struttura attraverso gli input forniti, senza che questi vengano etichettati in alcun modo.
- **Reinforcement Learning**. Il modello interagisce con un ambiente dinamico nel quale cerca di raggiungere un obiettivo (per esempio guidare un veicolo), avendo un insegnante che gli comunica solo se ha raggiunto lo scopo attraverso un meccanismo di premi e punizioni in base alla correttezza delle scelte intraprese.

Un particolare tipo di modelli di apprendimento automatico sono le cosiddette **reti neurali artificiali** (o **ANN**, ovvero **Artificial Neural Networks**), strutture costituite da molteplici strati di nodi elementari, che simulano in maniera semplificata il funzionamento del cervello umano. Infatti ogni nodo,

proprio come un neurone, è collegato ad altri nodi con i quali scambia input e output.

1.1 Le Reti Neurali Artificiali

Sebbene i tentativi di studio del cervello umano da parte dell'uomo siano vecchi migliaia di anni, il primo passo verso le reti neurali fu compiuto nel 1943, quando Warren McCulloch, psichiatra e neuro-anatomista, collaborò con il giovane matematico Walter Pitts nella scrittura del paper *Logical calculus of the ideas immanent in nervous activity* [1] in cui venne spiegato il funzionamento dei neuroni. I due studiosi modellarono una prima semplice rete neurale con l'utilizzo di circuiti elettrici. In particolare, un neurone biologico, somma i segnali di input provenienti dalle sinapsi che sono collegate ai dendriti di altri neuroni. Quando il segnale raggiunge una soglia limite il neurone genera un segnale di output verso altri neuroni.

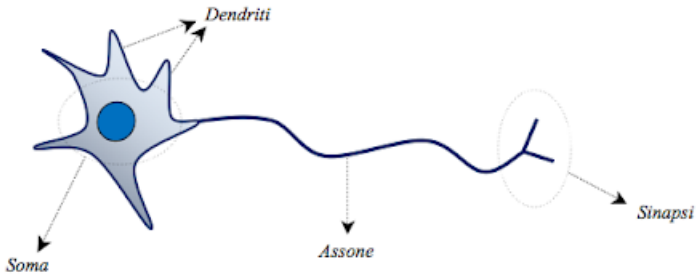


Figura 1.1: Rappresentazione stilizzata di un neurone biologico

Similmente, il funzionamento del neurone di McCulloch-Pitts prevede la somma di svariati input con pesi diversi fino all'innesco di una funzione di attivazione che porta alla produzione di un output, come si può osservare nella figura 1.2.

Nel 1949, Donald Hebb riprese il concetto di neurone nel suo libro *The Organization of Behavior* [2], dove espose come determinati processi neurali sono rafforzati ogni volta che vengono effettuati. Così, negli anni '50, i ricercatori cercarono di applicare questi studi ai sistemi computazionali fino a quando, nel 1954, il primo cosiddetto *Hebbian Network* fu implementato con successo all'MIT. Il passo successivo venne compiuto da Frank Rosenblatt, uno psicologo della *Cornell University* che, mentre stava studiando i sistemi decisionali presenti all'interno dell'occhio di una mosca, propose l'idea del **Perceptron** [3] nel 1958. Si tratta di un sistema con una semplice relazione di input-output che, basato sul neurone di McCulloch-Pitts, somma diversi input con diversi

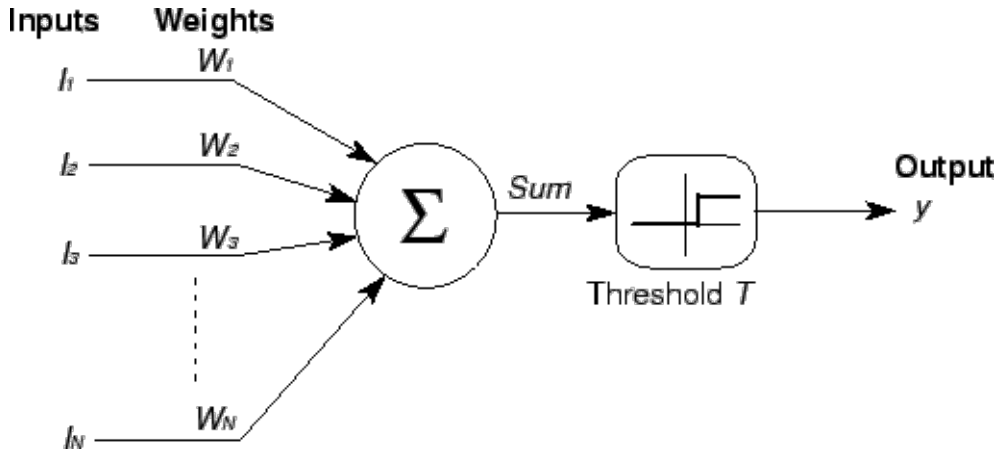


Figura 1.2: Rappresentazione del neurone di McCulloch-Pitts

pesi e, in caso il risultato sia maggiore di una determinata soglia restituisce 1, altrimenti 0. Il punto di forza di questo modello è che i pesi dei vari input vengono appresi attraverso input passati successivamente, in modo da ridurre al minimo la differenza tra l'output desiderato e quello effettivo. Questo modello, ancora alla base delle reti neurali artificiali attuali, permette di separare tra loro due classi ma con il limite che queste siano separabili linearmente. In quegli anni lo studio delle reti neurali artificiali subì una brusca accelerazione e, nel 1959, Bernard Widrow e Marcian Hoff svilupparono a *Stanford* il primo modello di rete neurale applicabile ad un problema reale. Questo sistema che prendeva il nome **MADALINE**, per via dell'utilizzo di *Multiple ADAptive LINear Elements*, e che permetteva di eliminare i rumori di disturbo dalle linee telefoniche è ancora oggi in uso. Come per ogni invenzione tecnologica della storia si ebbe un crande clamore per le reti neurali fino a quando, nel 1969, Marvin Minsky, fondatore del laboratorio di intelligenza artificiale dell'*MIT*, non pubblico il libro *Perceptrons* [4]. In questa pubblicazione Minsky sosteneva che l'approccio del Perceptron di Rosenblatt non poteva essere tradotto con successo in una rete neurale multi-livello, in quanto sarebbero stati necessari tempi lunghissimi, se non infiniti, per addestrare i pesi di neuroni sparsi su più livelli. Il risultato fu il più totale condizionamento della comunità scientifica internazionale e, per i successivi 10-12 anni, nessuno dei principali istituiti di ricerca mondiali decise di finanziare progetti che riguardassero le reti neurali artificiali. Era iniziata l'era del cosiddetto *AI winter*. La primavera arrivò solamente nel 1982 quando, alla *National Academy of Sciences*, Jon Hopefield presentò il paper passato alla storia come *Hopefield Net* [5] e contenente un nuovo approccio per creare dispositivi utili utilizzando le ANN. Inoltre, nello stesso anno, alla *US-Japan conference on Cooperative/Competitive Neural Networks*, il Giappone annunciò

di voler intraprendere seri sforzi sulle reti neurali e, un po' anche per la paura di restare indietro, gli Stati Uniti cominciarono a destinare cospicui fondi alla ricerca in questo campo. Così, l'*American Institute of Physics* stabilì nel 1985 il meeting annuale *Neural Networks in Computing* seguito dalla prima conferenza internazionale sulle reti neurali organizzata dall'*Institute of Electrical and Electronic Engineers (IEEE)* nel 1987. Si trattò comunque di una riscoperta di quei concetti che avevano iniziato a svilupparsi negli anni '60, come ad esempio la **backpropagation**, metodo basato sull'intuizione di attribuire un significato minore agli eventi cronologicamente precedenti, che, insieme alla discesa del gradiente, costituisce il cuore pulsante delle reti neurali moderne. Fu negli anni '90 che le reti neurali tornarono definitivamente, catturando l'attenzione del mondo intero e superandone le aspettative, a partire dallo sviluppo della **Long Short-Term Memory (LSTM)** [6] nel 1997 fino ad arrivare alla pubblicazione del paper *Attention is all you need* [7] sui **transformers** nel 2017.

1.2 Elaborazione del linguaggio naturale

L'**elaborazione del linguaggio naturale**, la cosiddetta **NLP** (natural language processing), è un campo dell'intelligenza artificiale nel quale il calcolatore analizza e comprende informazioni a partire dal linguaggio naturale, ovvero il linguaggio umano. Utilizzando NLP uno sviluppatore può organizzare e strutturare la conoscenza per eseguire automaticamente operazioni come la traduzione, il riconoscimento vocale, il sentiment analysis, la generazione di testo e molto altro ancora. Tuttavia si tratta di un campo molto complesso, infatti il linguaggio naturale è raramente preciso o chiaro (e.g. in caso di registrazione vocale) e presenta inoltre degli elementi di confusione che un sistema automatizzato fatica ad elaborare correttamente quali la sinonimia, la polisemia, l'ironia e le negazioni. Dunque, per comprendere il linguaggio umano, non è necessario comprendere solamente le parole ma anche i concetti che queste esprimono e come sono collegati tra loro per ottenere un significato. Sebbene il linguaggio sia una delle cose più semplici da apprendere per la mente umana, la sua ambiguità rende l'elaborazione del linguaggio naturale un compito molto difficile da assolvere per i calcolatori, i quali lavorano invece abitualmente con dati strutturati (e.g. dati in formato tabellare).

1.2.1 Le reti neurali in NLP

Con il termine **deep learning** si intende un insieme di tecniche basate su reti neurali artificiali organizzate in diversi strati, dove ogni strato calcola i

valori per quello successivo affinché l'informazione venga elaborata in maniera sempre più completa. Alcuni esempi sono i *deep neural networks (DNN)*, le *convolutional neural networks (CNN)* e le *recurrent neural networks (RNN)*. Negli ultimi anni diverse tecniche di deep learning sono state applicate con successo all'elaborazione del linguaggio naturale ottenendo risultati sorprendenti e, molte di esse, sono diventate lo stato dell'arte per alcuni dei principali task di NLP. L'esempio più classico riguarda la traduzione da una lingua all'altra, dove la *neural machine translation* ha sovraperformato totalmente altri metodi tradizionali come ad esempio la *statistical machine translation*. Un altro vantaggio della prima riguarda il fatto che i *deep neural networks* sono capaci di estrarre internamente e in maniera del tutto automatica le feature più opportune per il task offrendo così un'alta rappresentabilità del dato, mentre i tradizionali approcci di *statistical machine translation* richiedono un grande lavoro di feature engineering. Inoltre, con il deep learning, la rappresentazione di dati in diverse forme, come ad esempio testo e immagini, viene effettuata con vettori di numeri reali chiamati embeddings. Essi altro non sono che un punto in uno spazio multi-dimensionale, dove la distanza tra gli elementi ha un significato che un calcolatore può interpretare (per esempio, se si addestra il modello ad elaborare la semantica di una frase allora troveremo vicine frasi con un significato simile).

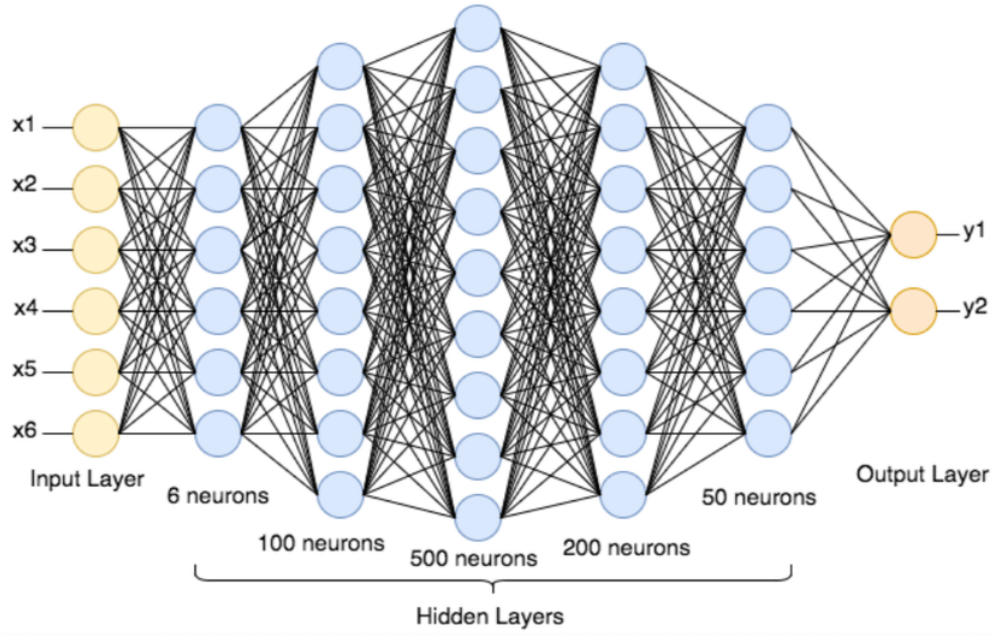


Figura 1.3: Rappresentazione di un deep neural network generico

1.3 Metric Learning e Deep Metric Learning

Per comprendere i concetti di metric learning e deep metric learning è necessaria una breve premessa sulla definizione di **metric**, ovvero una funzione tra due punti x e y come ad esempio $g(x, y)$ che ne descriva la distanza. Questa funzione deve essere non negativa, simmetrica e rispettare la disuguaglianza triangolare. Alcuni esempi di metrica in uno spazio vettoriale di dimensione d sono:

- La distanza euclidea:

$$d(x, y) = \sqrt{\sum_{i=1}^{d} (x_{\mathrm{i}} - y_{\mathrm{i}})^2}$$

- La distanza discreta:

$$d(x, y) = \begin{cases} 1 & \text{if } x \neq y, \\ 0 & \text{if } x = y \end{cases}$$

I basici algoritmi di machine learning hanno lo scopo di trovare una serie di regole o funzioni complesse che siano in grado di mappare gli input di un dato dataset nelle corrispondenti label di output (e.g. trovare una regola per stabilire se una recensione è positiva o negativa). Uno dei più semplici algoritmi di machine learning che fa uso del concetto di distanza è il **K-Nearest Neighbours (KNN)**, la cui idea è quella di individuare i k elementi più vicini ad un nuovo dato e assegnare ad esso la stessa label assegnata alla maggior parte di questi vicini. Similmente, l'obiettivo del **metric learning** è quello di apprendere una funzione di similarità dai dati. Mira infatti alla creazione di vettori di rappresentazione dei dati nei quali, e.g. i dati relativi a due fotografie del volto della stessa persona sono rappresentati vicino, mentre quelli relative a due fotografie di persone diverse sono rappresentati lontano. Quest'approccio si basa sulle metriche, che permettono di separare tra loro i dati in differenti classi, potendo così stabilire il grado si similarità tra diverse immagini. Tuttavia il metric learning conta alcune criticità, come il fatto che fotografie dei volti della stessa persona in diverse pose, con diverse espressioni o con diversa illuminazione possono facilmente ingannare un sistema di questo genere. Inoltre, vi sono solamente limitate capacità di attribuzione del grado di similarità in caso di non linearità dei dati. L'esigenza di superare questi limiti ha portato allo sviluppo del **deep metric learning**, che utilizza le reti neurali profonde per apprendere in maniera automatica feature discriminanti dalle immagini e solo successivamente calcolare le metriche. In questo modo si

risolve anche il problema di difficoltà di approccio alla non linearità dei dati. Queste tecniche sono state pensate principalmente per la *computer vision* e trovano la loro maggiore applicazione in campi come il riconoscimento facciale e l'individuazione di elementi tridimensionali in immagini e video.

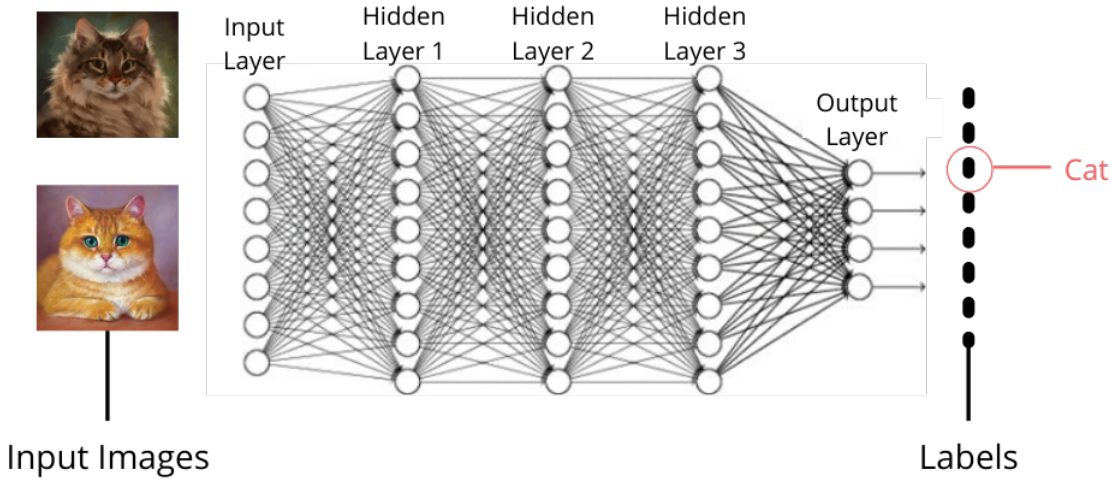


Figura 1.4: Rappresentazione di un deep neural network per la classificazione di immagini

1.4 Il progetto

Le reti neurali, per il loro corretto addestramento, necessitano di una X e della relativa y, i.e. di un'immagine e della sua classe (e.g. se si tratta di un cane o un gatto), di una frase e la sua flag (e.g. se è relativa ad un argomento o ad un altro), ecc. La y prende il nome di **label**, ovvero l'etichetta che caratterizza la X, i.e. l'input. Quindi, per il training delle reti neurali sono necessari dataset labellati, in cui a ciascun dato è associata la propria label (e.g. tabella in cui ad ogni recensione è associato un punteggio da 1 a 5). Tuttavia non tutti i domini, soprattutto quelli che riguardano il linguaggio naturale, sono rappresentati all'interno di un dataset labellato e crearne uno, nella maggior parte dei casi, sarebbe troppo dispendioso sia in termini di tempo che di risorse economiche, in quanto sarebbe necessario catalogare ad esempio una ad una tutte le frasi o i paragrafi di un documento. In questa tesi abbiamo deciso di utilizzare un dataset non labellato contenente documenti e pubblicazioni scientifiche sul **COVID-19** per creare un sistema di **information retrieval (IR)**. Lo scopo di un sistema di IR è quello di soddisfare il bisogno informativo di un utente, ovvero garantire a quest'ultimo, in seguito ad una sua interrogazione, i

documenti e le informazioni che ad essa corrispondono. In particolare, uno di questi sistemi è costituito da due elementi fondamentali:

- Le **query**, cosiddette interrogazioni, sono stringhe di parole chiave che rappresentano l'informazione richiesta. Vengono digitate dagli utenti (e.g. come avviene in un motore di ricerca, che è anch'esso un sistema di IR) e sono la concretizzazione del loro reale bisogno informativo.
- Gli **oggetti**, le entità che possiedono informazioni che potrebbero essere pertinenti alle interrogazioni degli utenti. Il sistema di IR si occupa quindi di restituire agli utenti gli oggetti attinenti alle query richieste.

Tuttavia creare questo sistema presenta i sopracitati problemi e non è quindi possibile addestrare alcun tipo di modello basato su reti neurali, in assenza di dati in linguaggio naturale non labellati.

1.4.1 Soluzioni esistenti

Lo stato dell'arte fornisce molte soluzioni per risolvere questo tipo di problemi, di seguito le principali:

- **Supervised Learning**. L'idea di base è quella di utilizzare qualche tipo di supervisore che possa guidare il modello nel processo di apprendimento automatico.
- **Weakly Supervised Learning**. In questo caso vengono utilizzate risorse limitate o imprecise per fornire un qualche tipo di segnale di supervisione all'addestramento che risulta appunto "debole".
- **Clustering**. Si tratta della più comune tecnica di apprendimento non supervisionato ed è in grado di rilevare similarità strutturali tra le osservazioni di un dataset, attraverso l'assegnazione di un insieme di osservazioni in sottogruppi, detti appunto cluster, di elementi tra loro omogenei.

Queste possibili soluzioni hanno tuttavia svariati limiti, per quanto riguarda le soluzioni supervised o weakly supervised risulta molto complesso riuscire a sviluppare modelli che ottengano risultati efficienti. Invece, relativamente al clustering, questo non permette di raggiungere gli stessi risultati delle reti neurali in termini di score.

1.4.2 L'idea chiave del progetto

Per risolvere questo genere di problemi, i.e. poter utilizzare le reti neurali su dataset non labellati, sono nate soluzioni di tipo **Self-Supervised**, che sfruttano cioè delle caratteristiche del dataset per addestrare un modello come **side**

effect (effetto collaterale) su uno specifico task. In particolare, il self-supervised learning è una sottocategoria dell'**unsupervised learning** in cui gli output e gli obiettivi sono definiti dal calcolatore che si occupa di labellare, categorizzare e analizzare autonomamente le informazioni traendo conclusioni basate su connessioni e correlazioni tra i dati. Abbiamo quindi messo a punto una soluzione di tipo self-supervised alla quale abbiamo applicato lo stato dell'arte del *natural language processing* e del *deep metric learning*. Ci siamo basati sull'assunzione che, se il deep metric learning, sviluppato per la computer vision, è in grado di "avvicinare" tra loro immagini simili e "allontanare" immagini differenti, dato che sia le immagini che il testo vengono rappresentati attraverso vettori di numeri reali, si possano utilizzare le stesse tecniche per "avvicinare" tra loro elementi testuali pertinenti (e.g. una query e un oggetto) e "allontanare" elementi non pertinenti. Abbiamo quindi sostituito il *convolutional neural network* tipicamente presente nei sistemi basati su deep metric learning con un modello di transformer, *ScientificBERT* [8], addestrato appositamente per compiere elaborazione del linguaggio naturale su documenti scientifici. Successivamente abbiamo testato diverse loss function caratteristiche del deep metric learning, mettendo in risalto le più performanti. Infine, abbiamo inserito i migliori language model ottenuti nel sistema di IR completo e li abbiamo confrontati con lo stato dell'arte attuale ottenendo risultati soddisfacenti.

Capitolo 2

Tecnologie disponibili

Per comprendere appieno il contesto di questo lavoro è necessario fare una panoramica sul dominio delle tecnologie a disposizione. Ricordiamo che il campo del natural language processing è in continua evoluzione e tutti i team di ricerca delle principali aziende globali del settore (e.g. Google) stanno dando il proprio contributo per la scoperta e la pubblicazione di modelli via via sempre più complessi ed efficienti.

2.1 Word Embeddings

Il termine **word embeddings** indica, nell'ambito del natural language processing, un insieme di tecniche di modellazione in cui parole o frasi di un vocabolario vengono mappate tipicamente in vettori di numeri reali. Secondo questo tipo di rappresentazioni parole con un significato simile vengono rappresentate in maniera simile. Ciò che più importa è dunque la semantica, che da modo al calcolatore, tramite queste rappresentazioni, di comprendere ed elaborare modelli di apprendimento automatico basati su testo in linguaggio naturale.

2.2 TF-IDF

Tf-idf [9] è una funzione di peso utilizzata per riconoscere l'importanza di un termine all'interno di un documento o di una collezione di documenti. Secondo tale funzione, il peso di un termine aumenta in maniera proporzionale al numero di volte in cui questo è contenuto all'interno del documento, ma cresce in maniera inversamente proporzionale alla frequenza del termine nella collezione. In questa maniera si conferisce più importanza ai termini che

appaiono con maggiore frequenza in un documento, ma che sono più rari all'interno dell'intera collezione.

2.2.1 Limiti del modello Bag of Words

Il modello **Bag of words** (BoW) [10] è una rappresentazione semplificata di documenti testuali utilizzata in natural language processing e in information retrieval. Questa rappresentazione prevede semplicemente di contare il numero di occorrenze di ogni termine presente all'interno di un documento e, in base a queste considerazioni, permette di scoprire quali termini sono più frequenti e dunque di maggior interesse.

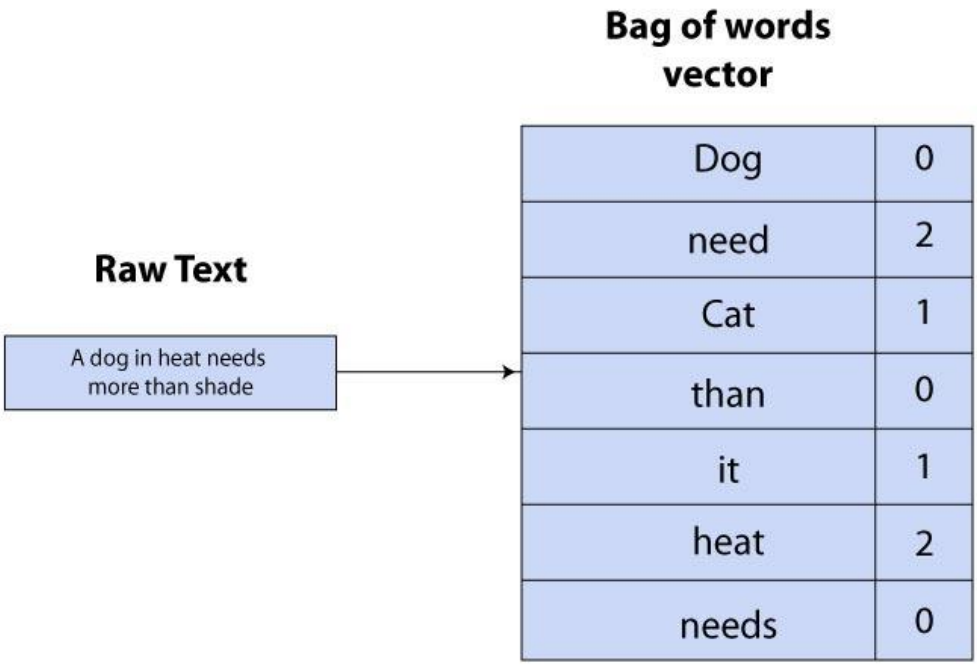


Figura 2.1: Esempio di rappresentazione BoW

Tuttavia questo modello ha diversi limiti non trascurabili. Consideriamo ad esempio la frase 'Oggi è una bella giornata'. Osserviamo che tutti i termini appaiono con frequenza uno ed è impossibile determinare attraverso una rappresentazione BoW perché l'aggettivo 'bella' sia maggiormente informativo rispetto al verbo 'è'. Può essere considerato un altro grande limite il fatto che

applicando, nella sua versione base, il modello ad un documento di 200 parole dove vi sono 20 occorrenze di 'il' e 15 di 'è' sia data maggiore importanza a questi termini privi di alcun contenuto informativo per i vari algoritmi di elaborazione del linguaggio naturale.

2.2.2 Vantaggi e formule di TF-IDF

Come sopra riportato, TF-IDF conferisce maggior importanza ai termini che compaiono frequentemente in un documento della collezione considerata e raramente negli altri. Così facendo si evita di dare importanza alle parole poco significative, le cosiddette **stopwords**, in quanto appaiono molto frequentemente in tutti i documenti.

Osserviamo ora il funzionamento di questa funzione, che può essere scomposta in due fattori:

- **TF - Term Frequency**. Indica la frequenza di un termine, ovvero il numero di apparizioni dello stesso all'interno di un documento. In genere questo numero viene diviso per la lunghezza del documento stesso evitando che siano così privilegiati i documenti più lunghi.
- **IDF - Inverse Document Frequency**. Questo fattore indica l'importanza generale di un termine all'interno della collezione. Si ottiene dividendo il numero di documenti presenti nella collezione considerata per il numero di documenti che contengono il termine preso in considerazione.

Scriviamo ora dunque la formula completa applicando un logaritmo in base 10 a ciascun fattore dato che l'importanza dei termini non cresce linearmente con la loro frequenza:

$$tf.idf(t,d) = log(f(t,d)) \cdot log(\frac{|Documenti|}{|d \in Documenti : t \in d|})$$

2.3 Okapi BM25

BM25 [11] è una funzione di ranking che stima la rilevanza di documenti rispetto ad una determinata query di ricerca. Questa restituisce una classifica in maniera indipendente dall'interrelazione dei termini all'interno di un documento, come ad esempio la loro vicinanza. Viene utilizzata dai motori di ricerca per ordinare i documenti in base ai termini inseriti e proporre dunque i risultati più pertinenti nelle prime posizioni. Troviamo spesso indicato *Okapi BM25* in quanto il sistema di information retrieval *Okapi* è stato il primo ad implementare queste funzionalità. *BM* significa invece *Best Match*.

2.3.1 Formule matematiche

Data una query Q contenente le keywords $q_1,....,q_n$ il punteggio BM25 di un documento D è:

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_\mathrm{i}) \times \frac{f(q_\mathrm{i},D)(k_\mathrm{i}+1)}{f(q_\mathrm{i},D) + k_1 \times (1 - b + b\frac{|D|}{avgdl})}$$

dove $f(q_\mathrm{i}, D)$ è la frequenza del termine q_i nel documento D, $|D|$ è la lunghezza del documento D in parole, *avgdl* è la lunghezza media dei documenti della collezione considerata, mentre k_1 e b sono iperparametri che assumono come valori di default rispettivamente $k_1 \in [1.2, 2.0]$ e $b = 0.75$. $IDF(q_\mathrm{i})$ è l'*inverse document frequency* rispetto al termine di ricerca q_i. Solitamente si calcola:

$$IDF(q_\mathrm{i}) = ln(\frac{N - n(q_\mathrm{i}) + 0.5}{n(q_\mathrm{i}) + 0.5} + 1)$$

dove N è il numero totale di documenti presenti nella collezione, mentre $n(q_i)$ è il numero di documenti contenenti q_i.

2.4 Singular Value Decomposition (SVD)

SVD [12] è una tecnica per la fattorizzazione di matrici propria dell'algebra lineare e basata sull'utilizzo di autovalori e autovettori. L'obiettivo è quello di creare un modello più complesso offline che sia poi in grado di fornire delle predizioni online in tempo costante. Questa tecnica è molto utile per la riduzione della dimensionalità, in quanto permette di individuare le features chiave per approssimare i dati e risulta inoltre molto scalabile.

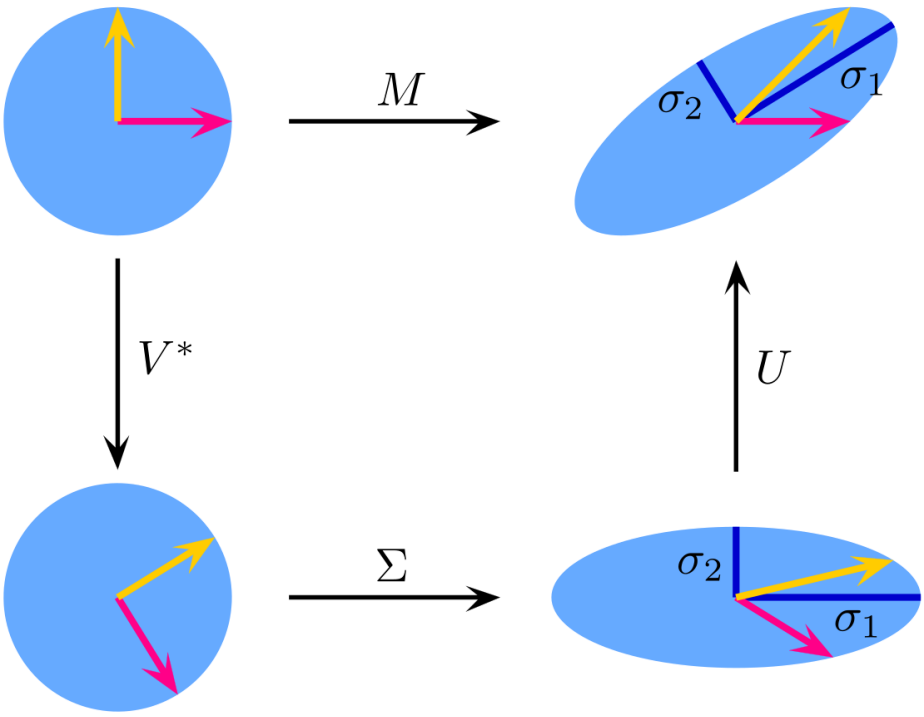
2.4.1 Funzionamento

Secondo la **Singular Value Decomoposition** una matrice M contenente dati può essere fattorizzata nel prodotto di tre metrici secondo la seguente formula:

$$M = U \times \Sigma \times V^\mathrm{T}$$

Dove è noto che:

- U contiene gli autovettori destri di $M \times M^{\mathrm{T}}$ e V gli autovettori sinistri di $M^{\mathrm{T}} \times M$.
- U e V sono due matrici unitarie, dunque le loro colonne sono ortonormali e costituiscono una base completa per un nuovo spazio in cui rappresentare i dati contenuti all'interno della matrice M. Vale inoltre:
  - $U \times U^{\mathrm{T}} = U^{\mathrm{T}} \times U = I$
  - $V \times V^{\mathrm{T}} = V^{\mathrm{T}} \times V = I$
- Le colonne di U e V sono ordinate per importanza decrescente del loro contenuto informativo, dunque u_1 e v_1 sono più importanti di u_2 e v_2
- Σ è una matrice diagonale non negativa che contiene gli autovalori di M disposti in ordine decrescente, dunque $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_{\mathrm{m}} \geq 0$.



$$M = U \cdot \Sigma \cdot V^*$$

Figura 2.2: Esempio di fattorizzazione SVD

Per generare un'approssimazione della matrice M si decide di considerare solamente i primi k autovalori presenti all'interno della matrice Σ, selezionando quindi solamente i dati più rilevanti per il problema in esame. Si è visto che con $k \in [10, 100]$ si ottiene solitamente una buona approssimazione. Questa fattorizzazione genera quindi un nuovo spazio a dimensionalità ridotta dove colloca i dati contenuti nella matrice M approssimandoli in base al valore di k scelto.

2.5 Latent Semantic Analysis (LSA)

La **Latent Semantic Analysis** [13] è una tecnica utilizzata nell'elaborazione del linguaggio naturale per stimare il significato semantico di un documento, ovvero la sua vicinanza ad un argomento o ad una materia particolare. Questa tecnica si basa sul fatto che ogni parola assume un significato particolare in un determinato contesto. Se ogni parola avesse un solo significato sarebbe molto semplice risalire alla semantica di una frase in maniera automatica, ma la realtà è ben più complessa, dato che nel linguaggio naturale un termine può assumere una miriade di accezioni creando ambiguità. Per risolvere questo problema è stata introdotta la Latent Semantic Analysis.

2.5.1 Funzionamento

La tecnica di Latent Semantic Analysis prevede l'estrazione di un insieme di documenti, dalla collezione oggetto di studio, che contengono una determinata parola chiave. Per fare ciò ogni singolo documento viene considerato come un vettore contenente le occorrenze di ogni termine (e.g. bag of word). Sono escluse dal conteggio le cosiddette *stop-word*, ovvero i termini più comuni e meno significativi per la semantica (e.g. congiunzioni e articoli); mentre vengono prese in considerazione le *index-word*, i.e. le parole che compaiono in almeno due documenti diversi. Successivamente, per la collezione di documenti, si crea una *matrice dei termini* contenente tutti i documenti costituenti la collezione e i relativi termini, con il conteggio delle occorrenze degli stessi all'interno di ogni documento.

Una volta ottenuta la matrice, la si fattorizza tramite **singular value decomposition**, riducendo così il numero di righe e preservando la similarità tra le colonne. I documenti vengono quindi confrontati a due a due sfruttando la similarità coseno:

$$sim\_coseno(a, b) = \frac{c_a \cdot c_b}{||c_a||_2||c_b||_2}$$

Dove a e b sono due documenti, mentre c_a e c_b le colonne della matrice dei termini contenenti rispettivamente il vettore dei termini del documento a e del documento b. I valori assunti dalla similarità coseno variano tra *0* e *1*. Più la similarità è vicina a *1* tanto più i due documenti presi in considerazione sono simili, mentre più il valore della similarità è vicino allo *0* tanto più i due documenti sono dissimili.

2.6 Word Vectors

I **Word Vectors** [14] rappresentano un significativo passo in avanti nella capacità di analizzare le relazioni tra parole, frasi e documenti. Grazie ad essi è possibile fornire ai calcolatori molte più informazioni sul testo in linguaggio naturale che utilizzando i sistemi di rappresentazione tradizionali. Quest'idea è inoltre alla base di tecnologie come la traduzione automatica e il riconoscimento vocale.

2.6.1 Limiti dei modelli di rappresentazione tradizionali

Gli approcci tradizionali all'elaborazione del linguaggio naturale, come ad esempio una rappresentazione di tipo *bag-of-word* o anche *tf-idf*, seppur utili per alcuni semplici task di machine learning non sono in grado di acquisire informazioni sul significato o il contesto di uno specifico termine. Ciò significa che alcune importanti relazioni, quelle di tipo sintattico-strutturale e quelle di tipo semantico, non vengono considerate. Dunque il linguaggio viene rappresentato in un modo che potremmo definire "ingenuo".

2.6.2 Vantaggi e funzionamento dei Word Vectors

Diversamente dagli approcci tradizionali, i Word Vectors sono vettori di numeri reali a virgola mobile, dove ogni elemento cattura una dimensione del significato di una parola. Dunque parole semanticamente simili sono rappresentate attraverso vettori simili. Essendo i vari termini trattati numericamente, possiamo applicare ad essi gli operatori matematici determinando una variazione del significato. L'esempio più comune è il seguente:

$$king - man + woman = queen$$

Ciò significa che possiamo sottrarre un significato dal vettore che rappresenta la parola *king*, i.e. la mascolinità, e sommarne un altro, i.e. la femminilità. In questo modo otteniamo un word vector che si avvicina molto a quello del

termine *queen*. Osserviamo che la semantica di una parola è quindi incorporata nelle dimensioni del vettore.

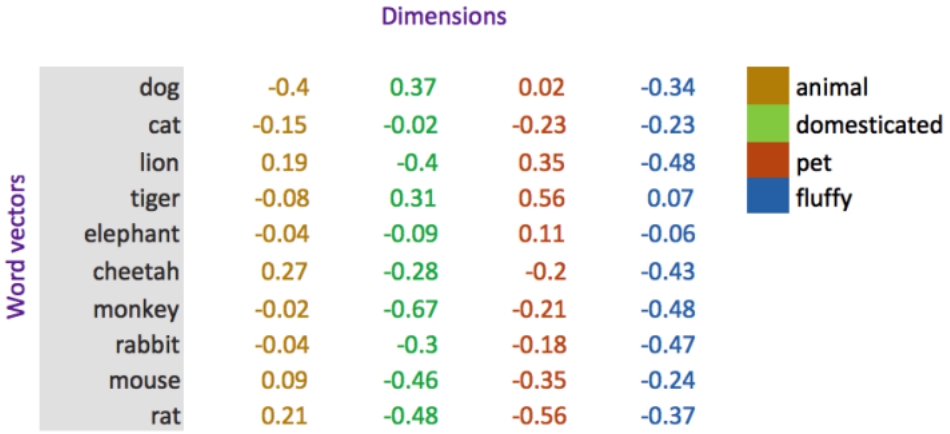


Figura 2.3: Esempio semplificato di rappresentazione tramite Word Vectors. In questa figura ogni dimensione cattura un significato preciso. Ad esempio la prima dimensione rappresenta il concetto di "animale", quindi il peso di ogni parola su quella dimensione determina quanto strettamente ci si riferisce a quel concetto

A questo punto sorge spontaneo chiedersi come si possano calcolare i valori reali contenuti nei word vectors. Esistono principalmente due diversi approcci:

- **Conteggio delle co-occorrenze di termini e contesti.** - Si basa sul fatto che i termini che condividono gli stessi valori di co-occorrenza all'interno di un documento abbiano un significato simile all'interno di uno specifico contesto. I valori vengono calcolati attraverso una matrice delle co-occorrenze, che viene spesso fattorizzata tramite SVD per ridurne la dimensionalità che sarebbe altrimenti molto elevata.

- **Previsione del contesto data la parola.** - Deriva dall'assunzione che parole che condividono contesti simili tendono ad avere significati simili. Il contesto di un termine si riferisce alle parole circostanti e i word vector sono tipicamente generati prevedendo la probabilità di un contesto data una parola. Quindi i pesi che compongono un word vector vengono appresi facendo previsioni sulla probabilità che altre parole siano contestualmente vicine alla parola data. Per fare ciò vengono tipicamente utilizzati modelli di reti neurali *skip-gram* [15] come *word2vec*.

2.7 Transformers

I **transformers** sono dei recenti modelli di deep learning introdotti per la prima volta nel 2017 da un team di ricercatori Google, attraverso il paper *Attention Is All You Need*, e utilizzati principalmente nell'elaborazione del linguaggio naturale. Adottano il meccanismo di attenzione e, come le *reti neurali ricorrenti (RNN)*, di cui sono la naturale evoluzione, sono progettati per gestire dati di input sequenziali (e.g. il linguaggio naturale) per attività come la traduzione e il question answering.

2.7.1 I modelli Seq2Seq

Per prima cosa definiamo **Seq2Seq** una particolare tipologia di modelli, noti principalmente nel campo della modellazione linguistica, che mira a trasformare una sequenza di input in una di output ed entrambe le sequenze possono essere di lunghezza arbitraria. Sono utilizzati per effettuare task come traduzione e question answering. Questi modelli hanno tipicamente un'architettura encoder-decoder composta da:

- **Encoder.** - Elabora le sequenze in input e le codifica in un *context vector* di lunghezza fissa. In questo modo si ha una rappresentazione efficace dei dati in ingresso.
- **Decoder.** - Riceve in ingresso il vettore modificato e decodifica il suo contenuto fornendo l'output desiderato.

Sia encoder che decoder sono Reti Neurali Ricorrenti e tipicamente utilizzano *LSTM (long-short term memory)*.

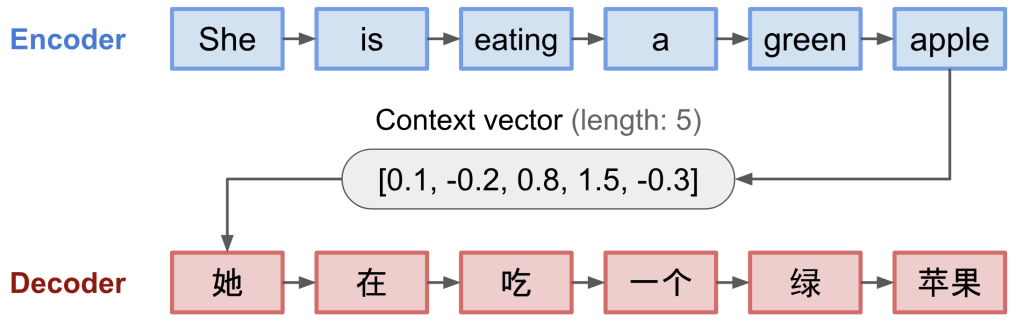


Figura 2.4: Esempio di Context Vector in un modello Seq2Seq di traduzione da inglese a cinese

Come si può evincere dall'immagine 2.4, il problema principale di un *context vector* a lunghezza fissa è l'incapacità di "ricordarsi" sequenze in input lunghe. Ad esempio viene "dimenticata" la prima parte una volta completata l'elaborazione dell'intero input e questo porta ad ottenere risultati che non esprimono al massimo il potenziale del modello.

2.7.2 Il meccanismo di attenzione

Prima di trattare in maniera approfondita i transformers è necessario fare una premessa sul meccanismo di attenzione che ne costituisce la base.

Le Reti Neurali Ricorrenti, che sono state per anni il culmine del progresso nell'elaborazione del linguaggio naturale, soffrono di due principali limiti. Il primo, come appena visto, è che tendono a "dimenticare" le informazioni più datate con il proseguire delle iterazioni, mentre il secondo è che non vi è alcun allineamento esplicito delle parole (e.g. in caso di traduzione da una lingua all'altra). Per superare questi limiti, è stato introdotto il meccanismo di attenzione applicato alle reti neurali, che ha trovato largo impiego nei task relativi alla traduzione da un linguaggio all'altro. L'idea di base del meccanismo è quella di creare un *context vector* che chiameremo c che rappresenti, per ogni dato di una sequenza in ingresso nel modello, la similarità con gli altri in modo da considerare tutte le relazioni latenti tra i vari dati. In questo modo il modello è in grado di capire su quali altre parti della sequenza in ingresso deve prestare "attenzione" in modo tale da ottenere un risultato il migliore possibile.

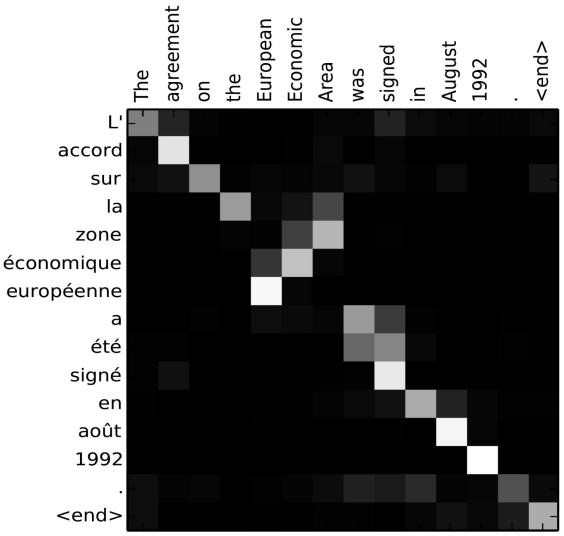


Figura 2.5: Rappresentazione grafica dell'attenzione durante una traduzione dal francese all'inglese

Per definire il meccanismo di attenzione in maniera più formale supponiamo di avere una sequenza in ingresso x di lunghezza n e una sequenza di output y di lunghezza m:

$$x = [x_1, x_2, ...x_{\mathrm{n}}]$$

$$y = [y_1, y_2, ...y_{\mathrm{n}}]$$

Definiamo l'encoder come una rete neurale ricorrente costituita dalla concatenazione di un *forward hidden state ($h_{i,f}$)* e di un *backward hidden state ($h_{i,b}$)*. Lo scopo di questa struttura è quello di includere nella propria rappresentazione le parole precedenti e successive a quella considerata.

$$h_{\mathrm{i}} = [{h_{\mathrm{i,f}}}^{\mathrm{T}}; {h_{\mathrm{i,b}}}^{\mathrm{T}}], i = 1, ..., n$$

La rete neurale del decoder ha come *hidden state* $s_{\mathrm{t}} = f(s_{\mathrm{t\text{-}1}}, y_{\mathrm{t\text{-}1}}, c_{\mathrm{t}})$ per la parola di output nella posizione t, dove $t = 1, ..., m$. Invece, il *context vector* c_t è la somma degli *hidden states* della sequenza di input, con pesi determinati dai punteggi di allineamento:

$$c_{\mathrm{t}} = \sum_{i=1}^{n} \alpha_{\mathrm{t,i}} h_{\mathrm{i}}$$

$$\alpha_{\mathrm{t,i}} = align(y_{\mathrm{t}}, x_{\mathrm{i}})$$

$$align(y_{\mathrm{t}}, x_{\mathrm{i}}) = \frac{e^{score(s_{\mathrm{t\text{-}1}}, h_{\mathrm{i}})}}{\sum_{i'=1}^{n} e^{score(s_{\mathrm{t\text{-}1}}, h_{\mathrm{i}'})}}$$

Dove $\alpha_{\mathrm{t,i}}$ indica quanto bene due parole y_t e x_i sono allineate. Calcoliamo tutto ciò attraverso alcuni punteggi di allineamento predefiniti. Secondo il paper di Bahdanau [16] il punteggio di allineamento α è parametrizzato da una rete *feed-forward* con un singolo *hidden layer*. Questa rete viene addestrata congiuntamente alle altre parti del modello. La forma della funzione di score è dunque la seguente, utilizzando *tanh* come funzione di attivazione non lineare.

$$score(s_{\mathrm{t}}, h_{\mathrm{i}}) = {v_{\mathrm{a}}}^{\mathrm{T}} tanh(W_{\mathrm{a}}[s_{\mathrm{t}}; h_{\mathrm{i}}])$$

Dove sia v_{a} che W_{a} sono entrambi matrici contenenti pesi che devono essere appresi dal modello. Queste matrici mostrano la correlazione tra le parole in input e quelle in output.

2.7.3 L'architettura di un Transformer

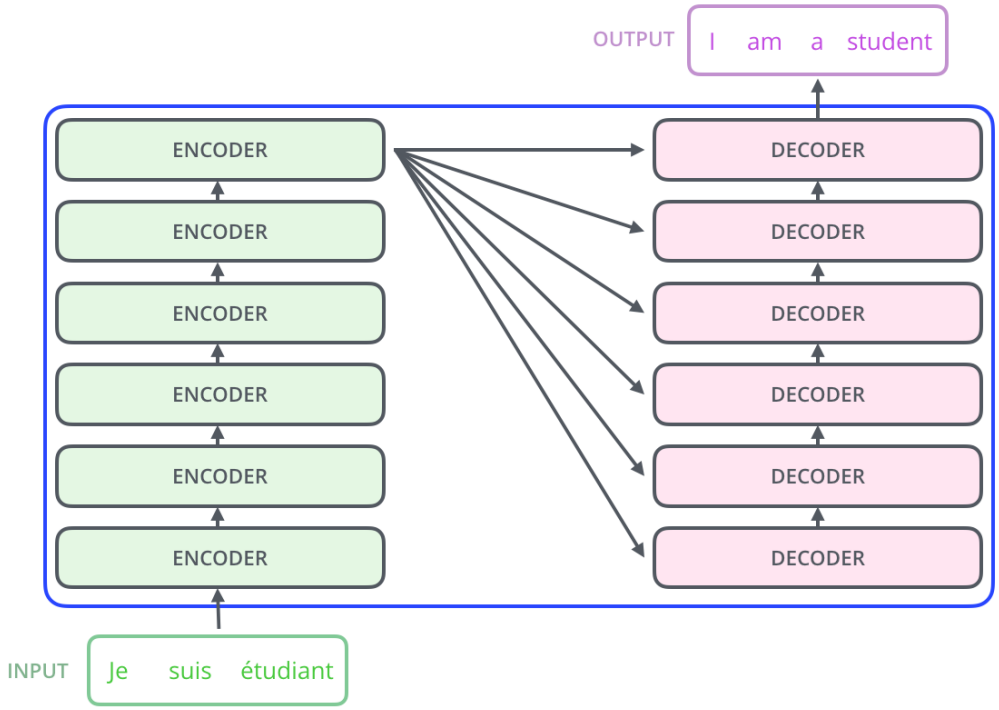


Figura 2.6: Rappresentazione grafica della struttura encoder-decoder di un transfomer

L'architettura dei **transformers** è di tipo encoder-decoder. L'encoder è composto da uno stack di $N=6$ layers, ognuno dei quali contiene due differenti sotto-layers: un **Multi-Head Self Attention Layer**, che come si può evnicere dal nome sfrutta il meccanismo di attenzione, e una rete neurale **Feed Forward**, costituita a sua volta da due layer e avente funzione di attivazione ReLu. L'output del primo sotto-layer viene combinato con il suo input per diventare l'input del secondo sotto-layer. L'output dell'intero layer costituisce poi l'input del layer successivo nello stack. Così come l'encoder, anche il decoder è composto da uno stack di $N=6$ layers. In questo caso ogni layer è costituito da tre diversi sotto-layer: due **Multi-Head Self Attention Layer** e una rete neurale **Feed Forward**. Successivamente allo stack del decoder vi è un **Linear Layer** che prende in input l'output del decoder e restituisce un output con dimensioni pari a quelle del dizionario del linguaggio naturale che si sta utilizzando. Come ultimo livello abbiamo il **SoftMax Layer** che genera la distribuzione di probabilità sul dizionario.

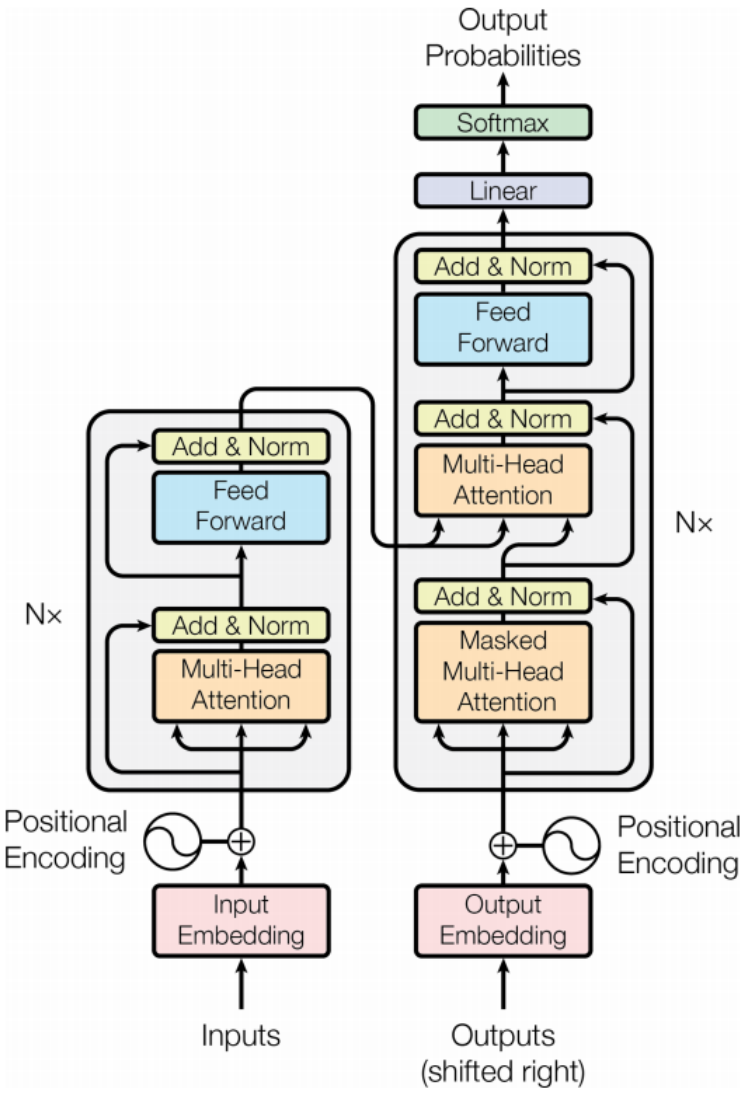


Figura 2.7: Rappresentazione grafica dell'architettura completa di un transfomer

2.7.4 Il meccanismo di attenzione nei transformers

Abbiamo visto che sia nell'encoder che nel decoder di un transformer sono presenti sei layer che contengono almeno una **Multi-Head Self Attention**. Procediamo quindi analizzando la **Self Attention** che si rifà al concetto di attenzione visto in precedenza. Supponiamo di avere in input la seguente frase da dover tradurre *"The animal didn't cross the street because it was too*

tired". Per noi esseri umani è piuttosto semplice capire che *"it"* è riferito ad *"animal"* e non a *"street"*, tuttavia non è altrettanto semplice per un algoritmo. Quindi, quando il modello sta processando la parola *"it"*, la self attenzion gli permette di poterlo associare direttamente ad *"animal"*. Per calcolare l'attenzione all'interno di un transformer abbiamo dunque bisogno di tre diverse matrici:

- **Q** - Matrice che contiene in ogni riga una diversa query q.
- **K** - Matrice che contiene in ogni riga una key k che rappresenta un valore.
- **V** - Matrice che contiene in ogni riga un vettore di valori v.

Una volta ottenute queste tre matrici calcoliamo la *self-attention* come mostrato di seguito nella figura 2.8, dove Q, K e V sono le matrici appena definite, mentre Z è la matrice di output contenente il risultato dell'operazione.

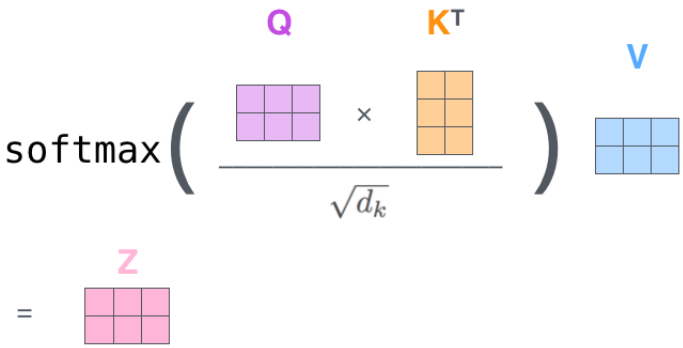


Figura 2.8: Formula per il calcolo dell'output di un self-attention layer

Ricordiamo tuttavia che l'architettura di un transformer non è costituita semplicemente da **Self Attention**, bensì da **Multi-Head Self Attention**. Nel paper "Attention is all you need" è stato infatti dimostrato che l'utilizzo di un numero superiore di attention heads, in particolare otto, permette di ottenere risultati migliori. Quindi l'attenzione viene calcolata più volte e alla fine ridimensionata per essere compatibile con il modello.

2.7.5 Input e output dei transformers

I transformers prendono in ingresso la rappresentazione delle parole di una frase tramite *input embeddings* e, per tenere traccia della posizione di una parola all'interno di una frase, utilizzano un particolare vettore chiamato *positional encoding*, che è posto al di sotto degli stack di encoder e decoder. Per calcolare

gli elementi di questo vettore si utilizzano le funzioni seno e coseno a diverse frequenze nella seguente maniera:

$$PE_{(pos,i)} = \begin{cases} sin(\frac{pos}{10000^{\frac{k}{d_{model}}}}) & \text{if } i = 2k, \\ cos(\frac{pos}{10000^{\frac{k}{d_{model}}}}) & \text{if } i = 2k+1 \end{cases}$$

Dunque tutte le posizioni pari sono calcolate utilizzando la funzione seno, mentre tutte quelle dispari utilizzando la funzione coseno. I *positional encoding* calcolati attraverso la formula vengono sommati ai vettori di embedding, in modo che ogni parola sia mappata con una frequenza diversa a seconda della sua posizione all'interno della frase. In questo modo il modello può ricavare le informazioni sulla posizione direttamente dai vettori di word encoding.

Per quanto riguarda invece l'output del transformer esso è costituito dalla distribuzione della probabilità di ogni parola sopra tutto il dizionario fornito al modello. Come già accennato, per calcolare questa probabilità vengono utilizzati un Linear Layer e una funzione SoftMax che prendono in ingresso l'output dello stack del decoder e restituiscono una probabilità.

2.8 Bidirectional Encoder Rapresentation from Transformer (BERT)

Nel 2019 un gruppo di ricercatori del team *Google AI Language* ha introdotto un nuovo modello per l'elaborazione del linguaggio naturale chiamato **BERT** [17], acronimo di **Bidirectional Encoder Rapresentation from Transformer**. A differenza degli altri recenti modelli dello stesso ambito, BERT è pre-addestrato su dati non etichettati ed è stato quindi forzato a recuperare informazioni dai *context* di sinistra e destra in tutti i layer. Può essere facilmente eseguito il fine-tuning aggiungendo un layer di output, per creare modelli che raggiungano lo stato dell'arte in molti task come il question answering.

2.8.1 Il modello

L'architettura di BERT è quella di un transformer bidirezionale multi-livello basato sull'encoder presentato nel paper *Attention is all you need* e visto nella sezione precedente. Chiamando il numero di layers L, il numero di hidden states H e il numero di attention heads A possiamo differenziare due principali modelli di BERT pre-addestrati: **BERT base** e **BERT large**. Il primo è caratterizzato da *L=12*, *H=768*, *A=12* e un numero totale di parametri pari

a *110M*; mentre il secondo da *L=24*, *H=1024*, *A=16* e un numero totale di parametri pari a *340M*. Entrambi i modelli sono stati pre-addestrati usando *BlackCorpus* (800 milioni di parole) e la versione inglese di *wikipedia* (2.5 miliardi di parole).

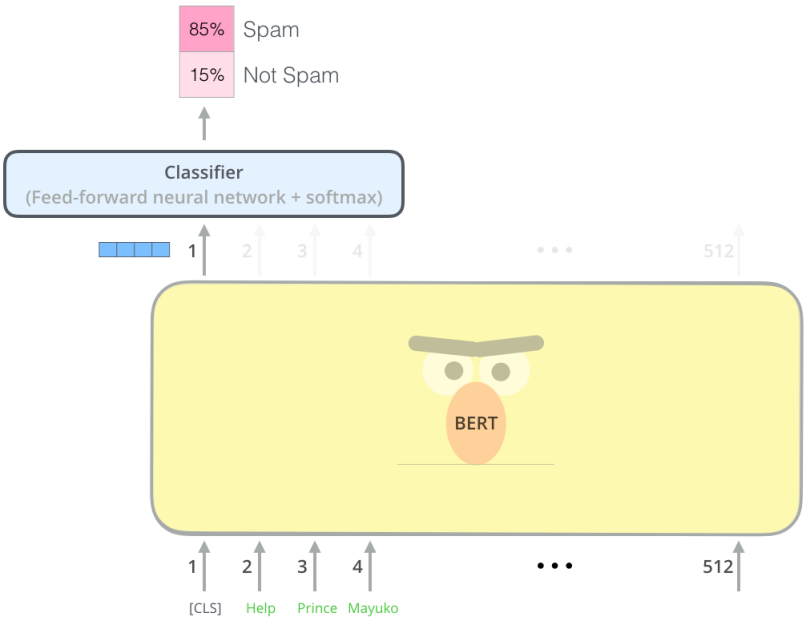


Figura 2.9: Esempio di modello BERT con l'aggiunta di un classification layer per classificare mail tra spam e non spam. All'interno del riquadro giallo sono da considerare i vari layer dell'encoder che costituisce BERT

2.8.2 Input e Output

Per essere ottimale in una grande varietà di task BERT è in grado di prendere in ingresso sia una singola frase che una coppia di frasi concatenate in un'unica sequenza di token. Per suddividere le frasi in token viene utilizzato *WordPiece*, un vocabolario contenente *30,000* token. Il primo token di ogni sequenza è sempre il token speciale di classificazione *[CLS]*, mentre viene utilizzato *[SEP]* per suddividere la prima frase dalla seconda nel caso ve ne siano due differenti in input. Inoltre, viene aggiunto un embedding appreso durante l'addestramento che indica se un determinato token appartiene alla frase in ingresso *A* o *B*. Quindi per un dato token, la sua rappresentazione è costituita dalla somma degli embedding del token, dei segmenti e delle posizioni.

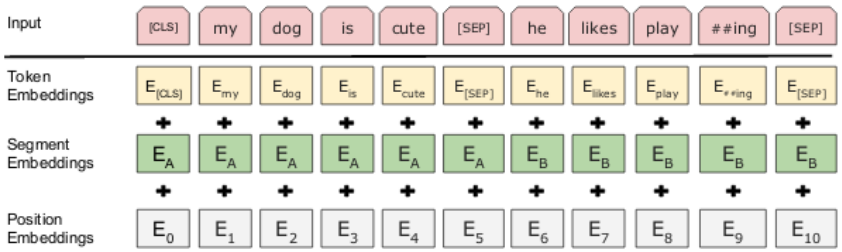


Figura 2.10: Schema della rappresentazione degli input di BERT

2.8.3 Performance

Al momento del suo rilascio BERT ha raggiunto lo stato dell'arte in numerosi task di elaborazione del linguaggio naturale, in particolare:

- **GLUE** (General Language Understanding Evaluation) [18] - Si è raggiunto lo stato dell'arte in ben nove differenti task di questo benchmark.
- **SQuAD** (Stanford Question Answering Dataset) [19] - Si è raggiunto lo stato dell'arte sia nelle versioni 1.1 e 2.0 di questo task in ambito question answering dell'università di Stanford.
- **SWAG** (Situations With Adversarial Generations) [20] - Si è raggiunto lo stato dell'arte su questo task basato su un dataset di domande a risposta multipla su situazioni di uso comune.

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
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Figura 2.11: Schema delle performance raggiunte da BERT base e BERT large sui principali task tel benchmark GLUE. Si può notare che lo score di BERT è maggiore rispetto a quello dei principali modelli precedentemente utilizzati

2.9 Scientific BERT (SciBERT)

Alcuni mesi dopo l'uscita di BERT i ricercatori dell'*Allen Institute for Artificial Intelligence* di Seattle hanno rilasciato **Scientific BERT**, una versione

di BERT pre-addestrata su paper scientifici. Dato che ottenere dati scientifici su larga scala per eseguire task di elaborazione del linguaggio naturale è complesso e molto costoso, si è pensato ad un modello che potesse colmare questa lacuna. Dunque l'addestramento non supervisionato di SciBERT su un grande corpus di documenti scientifici di alta qualità permette di risolvere in maniera efficiente diversi task di NLP a carattere scientifico.

2.9.1 Il modello

Per quanto riguarda il modello non ci sono particolari novità, in quanto si tratta di un'architettura identica a quella di BERT, a mutare è tuttavia il corpus sul quale SciBERT è stato pre-addestrato. In particolare la procedura di training è stata effettuata su tutti i paper scientifici del corpus di *semanticscholar.org*, costituito da 1.14 milioni di paper e 3.1 miliardi di token. I paper contenuti in questo corpus sono al 18% relativi al campo della computer science, metre il restante 82% è relativo al dominio biomedico.

2.9.2 Input e Output

Anche per quanto riguarda il funzionamento di input e output SciBERT mostra una struttura molto simile a quella del modello BERT originale. Osserviamo però che, mentre BERT utilizza *WordPiece* per la suddivisione delle frasi in token, i ricercatori dell'*Allen AI Institute* hanno creato *SciVocab*, un vocabolario bastato sempre su *WordPiece* e che utilizza la libreria *SentencePiece* per estrarre il contenuto dal corpus sul quale avviene l'addestramento. Le dimensioni del vocabolario sono poi impostate a 30,000, per corrsispondere con quelle del vocabolario base di BERT. L'efficacia dello *SciVocab* è superiore del 42% rispetto a quella del precedente, a dimostrazione del fatto che nelle pubblicazioni scientifiche sono maggiormente presenti termini scientifici rispetto a quelli di uso quotidiano contenuti nel *BaseVocab*.

2.9.3 Performance

In generale SciBERT sovraperforma BERT base per quanto riguarda i task a carattere scientifico. Si osserva un incremento dello score *F1* di +2.43 nel caso in cui non sia stato eseguito alcun fine tuning né su BERT base né su SciBERT, mentre di +2.11 nel caso in cui per entrambi sia stato eseguito il fine tuning. Possiamo quindi affermare con certezza che in caso di task specifici, il pre-addestramento di BERT su un dataset pertinente, in termini di argomento trattato, al task che si andrà ad eseguire porta ad un miglioramento degli score ottenuti. Di seguito si osserva la figura 2.12 con il confronto dei risultati

raggiunti tra BERT base e SciBERT sui principali task scientifici testati dagli autori del paper.

Field	Task	Dataset	SOTA	BERT-Base		SciBERT	
				Frozen	Finetune	Frozen	Finetune
Bio	NER	BC5CDR (Li et al., 2016)	88.85[7]	85.08	86.72	88.73	**90.01**
		JNLPBA (Collier and Kim, 2004)	**78.58**	74.05	76.09	75.77	77.28
		NCBI-disease (Dogan et al., 2014)	**89.36**	84.06	86.88	86.39	88.57
	PICO	EBM-NLP (Nye et al., 2018)	66.30	61.44	71.53	68.30	**72.28**
	DEP	GENIA (Kim et al., 2003) - LAS	**91.92**	90.22	90.33	90.36	90.43
		GENIA (Kim et al., 2003) - UAS	**92.84**	91.84	91.89	92.00	91.99
	REL	ChemProt (Kringelum et al., 2016)	76.68	68.21	79.14	75.03	**83.64**
CS	NER	SciERC (Luan et al., 2018)	64.20	63.58	65.24	65.77	**67.57**
	REL	SciERC (Luan et al., 2018)	n/a	72.74	78.71	75.25	**79.97**
	CLS	ACL-ARC (Jurgens et al., 2018)	67.9	62.04	63.91	60.74	**70.98**
Multi	CLS	Paper Field	n/a	63.64	65.37	64.38	**65.71**
		SciCite (Cohan et al., 2019)	84.0	84.31	84.85	**85.42**	**85.49**
Average				73.58	77.16	76.01	79.27

Figura 2.12: Schema delle performance raggiunte rispettivamente da BERT base e SciBERT sui principali task di NLP a carattere scientifico

2.10 Generative Pre-trained Transformer 3 (GPT-3)

GPT-3 [21], acronimo di **Generative Pre-trained Transformer 3**, è un modello per elaborazione del linguaggio naturale auto-regressivo che utilizza il deep learning per produrre testo simile a quello scritto da un essere umano ed eseguire altri task specifici di NLP. É stato sviluppato e rilasciato nel 2020 da *OpenAI*, un laboratorio di ricerca di San Francisco specializzato in intelligenza artificiale. A differenza degli esseri umani, che possono scrivere frasi di senso compiuto in una lingua a partire da pochi input o eseguire task con poche semplici istruzioni, per gli algoritmi di intelligenza artificiale sono necessarie decine di migliaia di esempi. Lo scopo di GPT-3 è quindi quello di ridurre questo divario tra algoritmi ed esseri umani per la risoluzione dei principali task di NLP.

2.10.1 Il modello

L'architettura di GPT-3 è molto simile a quella del suo predecessore GPT-2 [22] e si basa su un modello di transformer. La vera differenza è l'alternanza di *dense attention* e *locally banded sparse attention* nei diversi layer del transformer. Durante lo sviluppo di GPT-3 sono stati testati 8 differenti modelli di diverse

dimensioni, addestrati con un numero di parametri che spazia da 125 milioni a 175 miliardi.

Nella figura 2.13 sono mostrate l'architettura e le dimensioni dei diversi 8 modelli testati dal team di ricerca. In particolare, n_{params} è il numero totale dei parametri utilizzati per l'addestramento, n_{layers} è il numero totale di layer del transformer, d_{model} è la dimensione dei dati che il modello può processare contemporaneamente, n_{heads} è il numero di *attention heads*, mentre d_{head} la dimensione di ciascuna di esse. Inoltre, tutti i modelli utilizzano una *context window* di 2048 token e sono stati partizionati su più GPU per l'esecuzione dell'addestramento.

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Figura 2.13: Dimensioni, architettura e iperparametri dei vari modelli di GPT-3 addestrati da *OpenAI*

Tutti gli 8 modelli sono stati addestrati con 300 miliardi di token provenienti da una versione filtrata del dataset *Common Crawl*. In questo modo si è potuto ottimizzare il lavoro ed evitare l'addestramento sul dataset completo che è di proporzioni enormi. Basti pensare che, pur essendo stato addestrato solo su una parte filtrata di *Common Crawl*, all'epoca del suo rilascio GPT-3 era stato addestrato su una quantità di dati di 10 volte superiore a quanto mai fatto prima.

2.10.2 Performance

Il punto di forza del modello è il task di *text generation*, sul quale è stato addestrato. Infatti è utilizzato in *AI Dungeons* per creare giochi di avventura basati sul testo. Inoltre, l'ambizione di *Open AI* è stata pienamente soddisfatta, ovvero ottenere un modello in grado di eseguire qualsiasi task di NLP senza necessità di riprogrammazione. Viene anche in effetti utilizzato dalla *Microsoft* all'interno di alcuni programmi per tradurre linguaggio naturale in codice comprensibile ai calcolatori. Se volessimo fare un confronto con BERT potremmo dire che GPT-3 è stato addestrato su 175 miliardi di parametri, 470 volte in più di quelli su cui è stato addestrato BERT large. Questo garantisce

a GPT-3 di essere pronto ad affrontare diversi task basandosi su pochi semplici esempi, mentre BERT necessita di un fine-tuning massivo.

2.11 Deep Metric Learning

I recenti progressi nel deep learning hanno reso possibile l'apprendimento automatico della misura di similarità tra diverse immagini appartenenti ad uno specifico set. Per fare ciò si è utilizzato il **deep metric learning**, in grado di mappare immagini visivamente simili in posizioni vicine di un vettore di embeddings, mentre immagini visivamente dissimili su posizioni tra loro distanti. Tutto questo ha permesso di migliorare notevolmente i motori di ricerca basati sulle immagini.

Queste particolari reti vengono addestrate in modo molto simile alle comuni reti neurali in ambito deep learning, tuttavia utilizzano una tipologia di loss differenti, che sono appunto in grado di avvicinare immagini simili sullo spazio di rappresentazione durante l'addestramento. I pesi della rete vengono anche in questo caso ottimizzati con il supporto degli *optimizer*. Successivamente, durante l'*inference*, si estraggono le features dal layer di embedding e viene applicata una *nearest neighbor search* per identificare immagini simili a quella che si sta considerando.

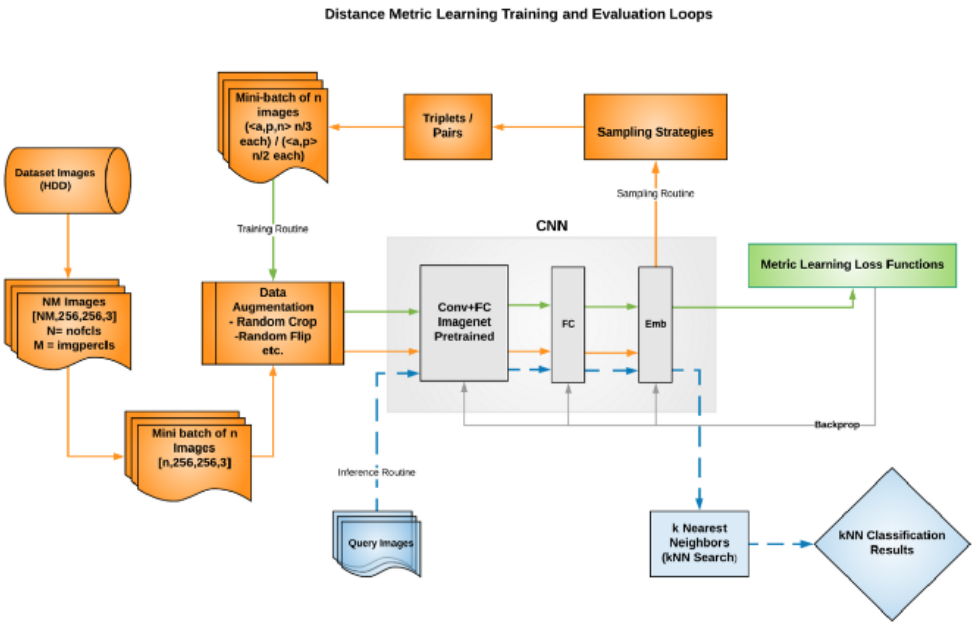


Figura 2.14: Schema di funzionamento di un modello di Deep Metric Learning

2.11.1 Contrastive Loss

Per minimizzare la distanza tra le feature di immagini simili e massimizzare la distanza tra le feature di immagini dissimili, la **contrastive loss** [23] codifica sia la misura della similarità che quella della dissimilarità indipendentemente l'una dall'altra se non sono separate da un dato margine chiamato α. La funzione di loss è quindi definita come segue:

$$L = \frac{1}{m} \sum_{(i,j)}^{\frac{m}{2}} y_{\mathrm{i,j}} D_{\mathrm{i^2,j}} + (1 - y_{\mathrm{i,j}})[\alpha - D_{\mathrm{i,j^2}}]^+$$

Dove m è il numero di coppie di immagini simili presenti in un mini-batch, $D_{\mathrm{i,j}} = ||f(x_{\mathrm{i}}) - f(x_{\mathrm{j}})||^2$ è la distanza tra le feature $f(x_{\mathrm{i}})$ e $f(x_{\mathrm{j}})$ che corrispondono rispettivamente alle immagini $\mathrm{x_i}$ e $\mathrm{x_j}$, $y_{\mathrm{i,y}} = +/-1$ indica se una coppia $(x_{\mathrm{i}}, x_{\mathrm{j}})$ condivide o meno una label simile. Inoltre $[\alpha - D_{\mathrm{i,j^2}}]^+$ è il risultato della funzione $max(0, [\alpha - D_{\mathrm{i,j^2}}]^+)$.

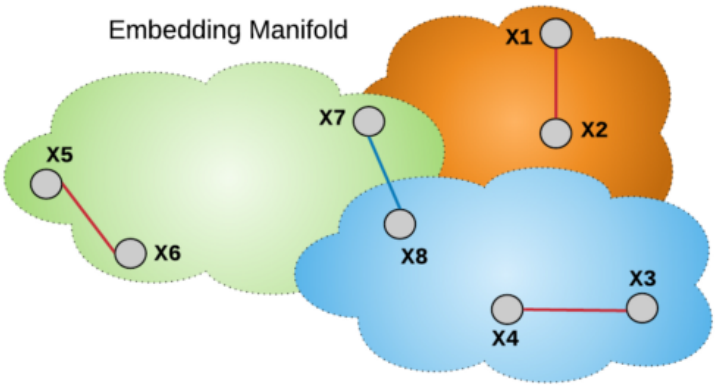


Figura 2.15: Rappresentazione di contrastive loss. I collegamenti rossi rappresentano coppie di immagini simili, mentre quelli blu coppie di immagini con label differenti

Sebbene la *contrastive loss* sia molto intuitiva da comprendere vi sono alcuni limiti dovuti alla presenza del margine costante α. In questo modo, per forza di cose, la loss non tiene conto della scala di dissomiglianza visiva per quanto riguarda tutti gli esempi negativi. Ciò limita lo spazio di embedding a registrare, durante l'addestramento, solo un sottogruppo di tutte le possibili dissimilarità e a disporre immagini diverse come se fossero simili tra loro.

2.11.2 Triplet Loss

La più nota e utilizzata funzione di loss in ambito *deep metric learning* è certamente la **triplet loss** [24]. Si basa su una tripletta di feature, $(x_{\text{ia}}, x_{\text{ip}}, x_{\text{in}})$ dove, $(x_{\text{ia}}, x_{\text{ip}})$ hanno label simili, mentre $(x_{\text{ia}}, x_{\text{in}})$ hanno label differenti. Supponendo di chiamare (x_{ia}) *anchor* (ancora), la distanza $D_{\text{ia,ip}}$ (cosiddetta *anchor-positive*) deve essere inferiore alla distanza $D_{\text{ia,in}}$ (cosiddetta *anchor-negative*) di almeno un dato margine α. Ecco di seguito la funzione completa:

$$L = \frac{3}{2m} \sum_{i}^{\frac{m}{3}} [{D_{\text{ia,ip}}}^2 - {D_{\text{ia,in}}}^2 + \alpha]^+$$

Dove $D_{\text{ia,ip}} = ||f(x_{\text{ia}}) - f(x_{\text{ip}})||^2$, $D_{\text{ia,in}} = ||f(x_{\text{ia}}) - f(x_{\text{in}})||^2$ e α è l'iperparametro relativo alla differenza minima tra le distanze *anchor-positive* e *anchor-negative*.

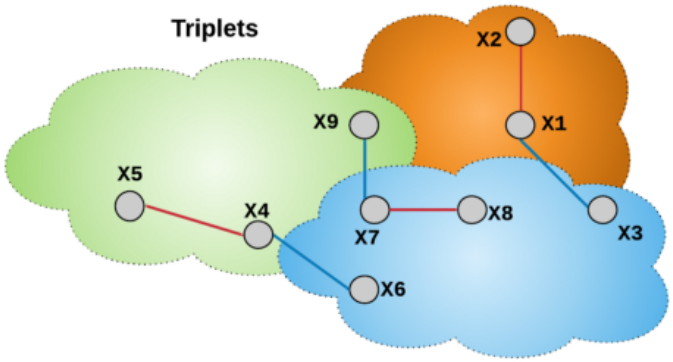


Figura 2.16: Rappresentazione di triplet loss. Ad esempio nella tripletta (x_1, x_2, x_3), (x_1) rappresenta l'ancora, che è connessa con la linea rossa all'elemento positivo (x_2) e con la linea blu all'elemento negativo (x_3)

Dato che il margine α nella loss non determina direttamente la distanza tra due embedding, ma la distanza tra le coppie all'interno della tripletta rispetto all'ancora, la triplet loss, a differenza della contrastive loss, tiene conto di tutte le possibili dissimilarità. Tuttavia, questa loss risulta computazionalmente molto costosa. Infatti, all'interno di un mini-batch, è necessario esplorare tutte le possibili combinazioni di triplette portando il costo computazionale ad un valore di $O(N^3)$. Dunque, per affrontare questa convergenza lenta, vengono spesso utilizzati appricci di tipo *mining-based*.

2.11.3 Lifted Structure Loss

Durante l'addestramento di una rete neurale con *triplet loss* non si riescono a sfruttare tutte le informazioni presenti nel mini-batch, principalmente per il fatto che la loss permette di analizzare solamente le due distanze *anchor-positive* e *anchor-negative* per ogni ancora. L'idea alla base della **lifted structure loss** [25] è invece quella di utilizzare tutte le coppie disponibili all'interno del mini-batch che si sta considerando in modo da ottenere, all'interno di una matrice, tutte le possibili distanze tra le varie coppie. Dunque, come si può evincere dalla figura 2.17, i negativi non vengono definiti utilizzando solamente l'ancora predefinita, ma anche l'elemento positivo viene a sua volta considerato come ancora per trovare gli elementi a lui negativi. Quindi entrambi i nodi di un sistema costituito da ancora ed elemento positivo vengono collegati a tutti gli elementi a loro negativi.

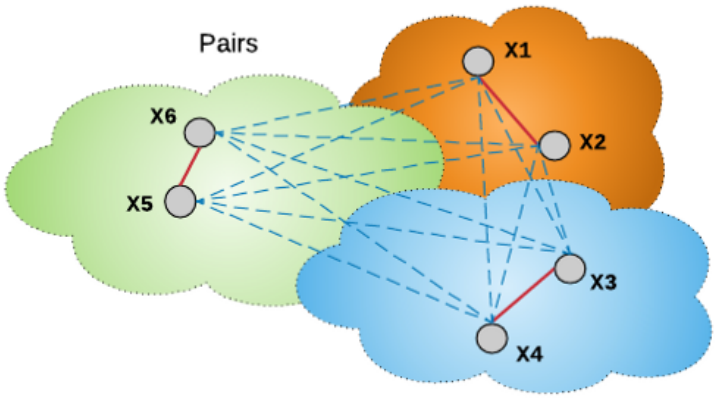


Figura 2.17: Rappresentazione di lifted structure loss. Per ogni coppia ancora-elemento positivo, come ad esempio (x_1, x_2), osserviamo che questa loss considera le interazioni di entrambi i membri della coppia con tutti gli altri elementi del mini-batch

Tuttavia, come si può immaginare, con questo approccio la velocità di convergenza risulta molto lenta, essendo necessario considerare tutte le possibili combinazioni tra i vari elementi del mini-batch. Per velocizzare il procedimento, gli autori della loss suggeriscono di utilizzare approcci di tipo *mining-based* per estrarre, per ognuno dei due elementi di ogni coppia ancora-elemento positivo il suo rispettivo elemento cosiddetto *hard negative*, ovvero quello che più gli è dissimile.

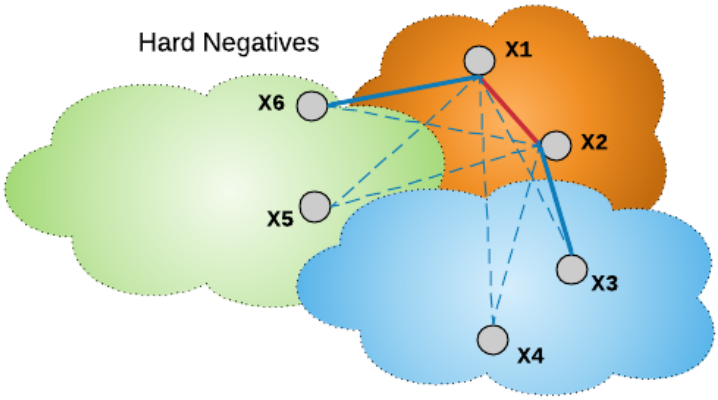


Figura 2.18: Rappresentazione di lifted structure loss con hard-negatives. Per la coppia ancora-elemento positivo (x_1, x_2) osserviamo che ognuno dei due elementi è collegato attraverso la linea spessa blu al suo hard-negative

Procediamo ora scrivendo la formula matematica della funzione lifted structure loss:

$$L_{\mathrm{i,j}} = log(\sum_{(i,k)\in N} e^{\alpha - D_{\mathrm{i,k}}} + \sum_{(i,l)\in N} e^{\alpha - D_{\mathrm{j,l}}} + D_{\mathrm{i,j}})$$

$$L = \frac{1}{2|P|} \sum_{(i,j)\in P} max(0, L_{\mathrm{i,j}})^2$$

Dove $D_{\mathrm{i,j}} = ||f(x_{\mathrm{i}}) - f(x_{\mathrm{j}})||^2$, mentre P e N sono il numero di coppie rispettivamente positive e negative presenti all'interno del mini-batch preso in considerazione.

2.11.4 Angular Loss

I tipi di loss visti finora sono accomunati dal focus che pongono sull'ottimizzazione di distanze assolute o relative. Contrariamente, l'**angular loss** [26] si basa sulla creazione di relazioni di terzo ordine in modo da ottenere distanze angolari. Per spiegare meglio dobbiamo considerare una tripletta, come quelle viste nella *triplet loss*, e immaginare di tracciare un triangolo tra i tre nodi che la costituiscono. Le relazioni di terzo ordine che vengono codificate saranno dunque relative agli angoli del triangolo considerato. In questo caso si considera quindi una distanza angolare, ovvero il coseno dell'angolo.

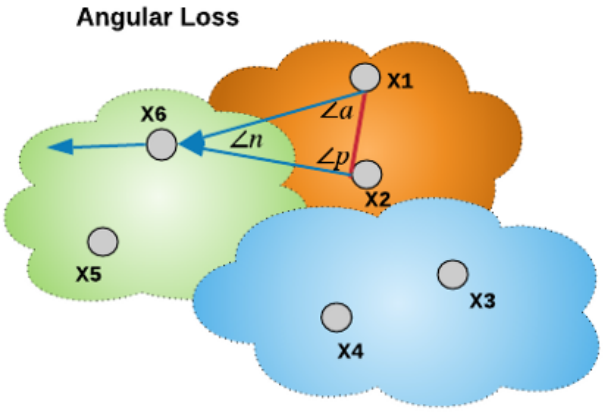


Figura 2.19: Rappresentazione di angular loss. Si può osservare il triangolo formato dalla tripletta (x_1, x_2, x_6), dove x_1 è l'ancora, x_2 l'elemento positivo e x_6 quello negativo

La formula matematica di questa funzione risulta quindi:

$$f_{\mathrm{a,p,n}} = 4tan^2\alpha(x_{\mathrm{a}} + x_{\mathrm{p}})^{\mathrm{T}}x_{\mathrm{n}} - 2(1 + tan^2\alpha){x_{\mathrm{a}}}^{\mathrm{T}}x_{\mathrm{p}}$$

$$L(\beta) = \frac{1}{N}\sum_{x_{\mathrm{a}}\in\beta} log(1 + \sum_{x_{\mathrm{n}}\in\beta, y_{\mathrm{n}}\neq y_{\mathrm{a}}, y_{\mathrm{p}}} e^{f_{\mathrm{a,p,n}}})$$

Dove a, p e n sono rispettivamente ancora, nodo positivo e nodo negativo, mentre α è un iperparametro che permette di regolare l'angolo da considerare specificandolo in gradi (nel paper vengono usati valori compresi tra 36 e 55). Una delle peculiarità dell'angular loss è che può essere combinata con le altre funzioni di loss migliorando le performance complessive. Inoltre, la geometria angolare permette di rendere i termini della loss molto più robusti rispetto all'utilizzo di una distanza euclidea, migliorando così l'efficacia degli addestramenti.

2.11.5 Multi-Similarity Loss

Prima di trattare in maniera specifica la **Multi-Similarity Loss** [27] è necessario fare una premessa sulle similarità su cui essa si basa. Eccone di seguito l'elenco:

- **Self Similarity** - Si calcola a partire dalle coppie di elementi considerate ed è la più importante per il computo della multi-similarity loss. Una

coppia ancora-elemento negativo con un'alta similarità coseno significa che è più difficile distinguere le classi dei due elementi. Queste coppie sono spesso chiamate *hard negative pairs* e sono molto più informative per il modello, in quanto lo rendono in grado di apprendere features più specifiche e discriminanti. Ovviamente, solamente con la self similarity è difficile descrivere totalmente la distribuzione di un mini-batch nello spazio di embedding e, inoltre, le correlazioni delle altre coppie possono avere un impatto significativo sulla computazione di questa similarità. Per questo si introducono similarità relative considerando tutte le coppie che hanno la stessa ancora della coppia considerata.

- **Negative Relative Similarity** - Si calcola la similarità di una singola coppia ancora-elemento negativo con tutte le altre coppie ancora-elemento negativo del mini-batch considerato aventi stessa ancora. Questa similarità cresce mano a mano che gli elementi negativi si avvicinano all'ancora.

- **Positive Relative Similarity** - Si calcola la similarità di una singola coppia ancora-elemento positivo con tutte le altre coppie ancora-elemento positivo del mini-batch considerato aventi stessa ancora. Questa similarità decresce mano a mano che gli elementi positivi diventano più vicini all'ancora.

Gli autori del paper suggeriscono di utilizzare solamente *hard negative pairs* e *hard positive pairs* per l'addestramento e di scartare tutte le altre coppie in quanto o limitano un possibile aumento di performance o tendono addirittura a peggiorarle. Scegliendo quindi solo le coppie che contengono il maggior contenuto informativo si può abbassare anche il costo computazionale complessivo dell'algoritmo. Per selezionare quindi solamente gli elementi di interesse si utilizzano tecniche di mining. In particolare, in questo caso sono:

- **Hard Negative Mining** - Si tratta di filtrare i negativi la cui similarità con l'ancora è maggiore della similarità minima dei positivi, ovvero di quella del positivo che si trova più lontano dall'ancora nello spazio di embeddings. Di seguito la formula:

$$S_{\mathrm{ij}}^{-} > min_{y_{\mathrm{k}}=y_{\mathrm{i}}} S_{\mathrm{ik}} - \epsilon$$

- **Hard Positive Mining** - Si tratta di filtrare i positivi la cui similarità con l'ancora è minore della massima similarità dei negativi, ovvero di quella del negativo che si trova più vicino all'ancora nello spazio di embeddings. Di seguito la formula:

$$S_{\mathrm{ij}}^{+} < max_{y_{\mathrm{k}}\neq y_{\mathrm{i}}} S_{\mathrm{ik}} + \epsilon$$

A questo punto è possibile enunciare la formula completa della *multi-similarity loss*:

$$L = \frac{1}{m}\sum_{i=1}^{m}\left\{\frac{1}{\alpha}log\left[1+\sum_{k\in P_{\mathrm{i}}}e^{-\alpha(S_{\mathrm{ik}}-\lambda)}\right]+\frac{1}{\beta}log\left[1+\sum_{k\in N_{\mathrm{i}}}e^{\beta(S_{\mathrm{ik}}-\lambda)}\right]\right\}$$

Dove S_{ik} è la similarità tra le coppie, λ è il margine di similarità, mentre α e β sono iperparametri.

L'utilizzo combinato di tre diverse similarità e dell'*hard pair mining* ha permesso alla *multi-similarity loss*, al momento della sua pubblicazione, di divenire lo stato dell'arte relativo al *deep metric learning*.

Capitolo 3

Overview del progetto

La COVID-19, nota anche come malattia da coronavirus 2019, è una malattia infettiva respiratoria acuta causata dal virus denominato SARS-CoV-2 e appartenente alla famiglia dei coronavirus. Dopo lo sviluppo dei primi casi di questo nuovo coronavirus in Cina nell'autunno del 2019, ad inizio 2020 l'intero pianeta è precipitato in una pandemia globale tuttora in corso, che ha stravolto le nostre vite con conseguenze che non si vivevano dall'influenza spagnola a cavallo tra gli anni '10 e gli anni '20 del secolo scorso. La comunità scientifica internazionale si è immediatamente mobilitata per trovare trattamenti efficaci e vaccini contro questo virus e, fortunatamente, a differenza di 100 anni fa, oggi è disponibile l'importante contributo della computer science e, in particolare, dell'intelligenza artificiale e dell'information retrieval, grazie alle quali è possibile reperire in modo rapido quante più informazioni possibili a partire dagli studi scientifici del passato su malattie precedenti o simili, cercando in questo modo di velocizzare la reazione delle autorità sanitarie globali per fronteggiare questo grave dramma. Questo progetto studia nuovi approcci self-supervised per l'addestramento di metric neural networks da utilizzare in sistemi di Information Retrieval aventi come dominio la letteratura sul COVID19. In particolare, abbiamo proposto una comparazione formale delle più recenti soluzioni di Deep Metric Learning, applicate in un contesto privo di dati labellati e abbiamo quindi cercato di capire quali di esse si sono adattate meglio a questo tipo di situazione. Infine abbiamo costruito un sistema IR completo capace di selezionare paper semanticamente simili a una query espressa tramite linguaggio naturale da un utente, e lo abbiamo confrontato con lo stato dell'arte sull'evaluation set TREC-COVID.

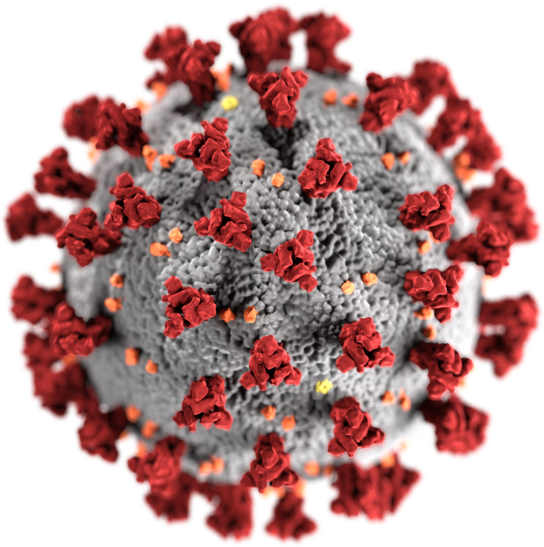


Figura 3.1: Rappresentazione grafica del coronavirus SARS-CoV-2

3.1 CORD-19: The Covid-19 Open Research Dataset

La prima versione di *CORD-19: The Covid-19 Open Research Dataset* [28], il dataset utilizzato per questo progetto, è stata rilasciata il 16 marzo 2020, sotto il coordinamento del *Georgetown University's Center for Security and Emerging Technology* (CSET) e con il contributo di *Allen Institute for AI* (AI2), *The White House Office of Science and Technology Policy* (OSTP), *The National Library of Medicine* (NLM), *The Chan Zuckerburg Initiative* (CZI), *Microsoft Research* e *Kaggle*. Questo dataset è una grande collezione di pubblicazioni scientifiche sul Covid-19 e altri coronavirus storicamente correlati ad esso come la SARS e la MERS. Inizialmente era costituito approssimativamente da 28.000 paper ed in poche settimane è cresciuto di altre decine di migliaia. Viene continuamente aggiornato e, secondo gli autori, si proseguirà in questa direzione fino alla fine della crisi causata dalla pandemia. CORD-19 mira a connettere tra loro diverse comunità, quella relativa al machine learning, quella di esperti del settore biomedico e quella degli amministratori, nella corsa contro il tempo per identificare trattamenti efficaci e politiche di gestione della crisi Covid-19. L'obiettivo è quello di sfruttare questi pool di conoscenze diversificate e complementari per scoprire il più rapidamente possibile informazioni utili a partire dalla letteratura disponibile a riguardo. Chi sta sfruttando questo dataset sta applicando e studiando tecniche basate sull'intelligenza artificiale

Tabella 3.1: Distribuzione delle categorie dei paper

Categoria	**% di CORD-19**
Virologia	42.3%
Immunologia	20.7%
Biologia Molecolare	12.7%
Genetica	8.0%
Medicina intensiva	6.7%
Altro	9.6%

per information retrieval e natural language processing in modo da estrarre efficacemente nozioni cruciali.

CORD-19 integra paper provenienti da diverse fonti (come si può osservare dalla figura 3.2). Ogni paper è definito come l'unità di base della conoscenza pubblicata ed è associato ad un insieme di metadati, come titolo, bibliografia, autori, luogo e data di pubblicazione.

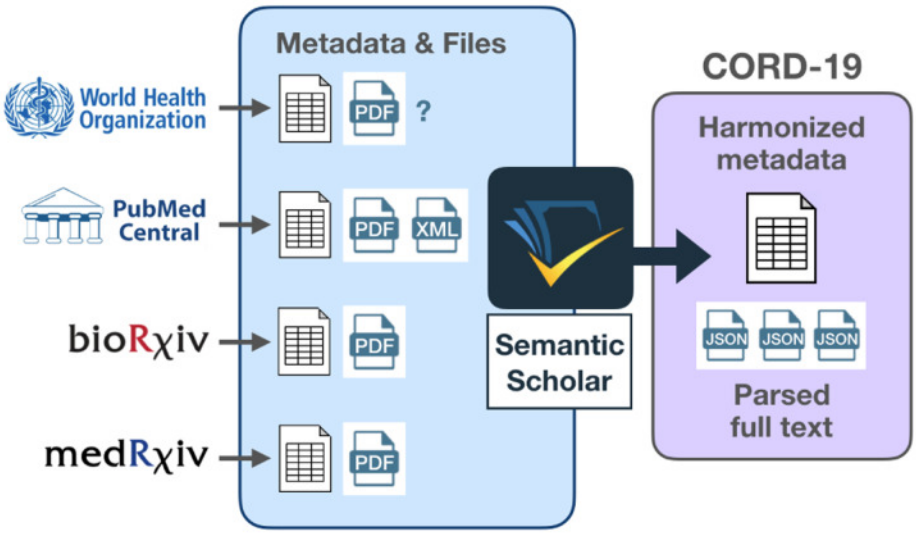


Figura 3.2: Fonti dei documenti presenti in CORD-19. I paper sono collezionati da Semantic Scholar e vengono poi processati per estrarne il testo completo

I documenti provenienti dalle diverse fonti sono raccolti da *Semantic Scholar*, che provvede ad armonizzare e deduplicare i metadati in modo tale da processare i dati per estrarne il testo completo. Vengono inseriti nel dataset i soli documenti che permettono l'accesso libero e la diffusione senza violazione del copyright (e.g. documenti con licenza *Creative Commons* (CC)).

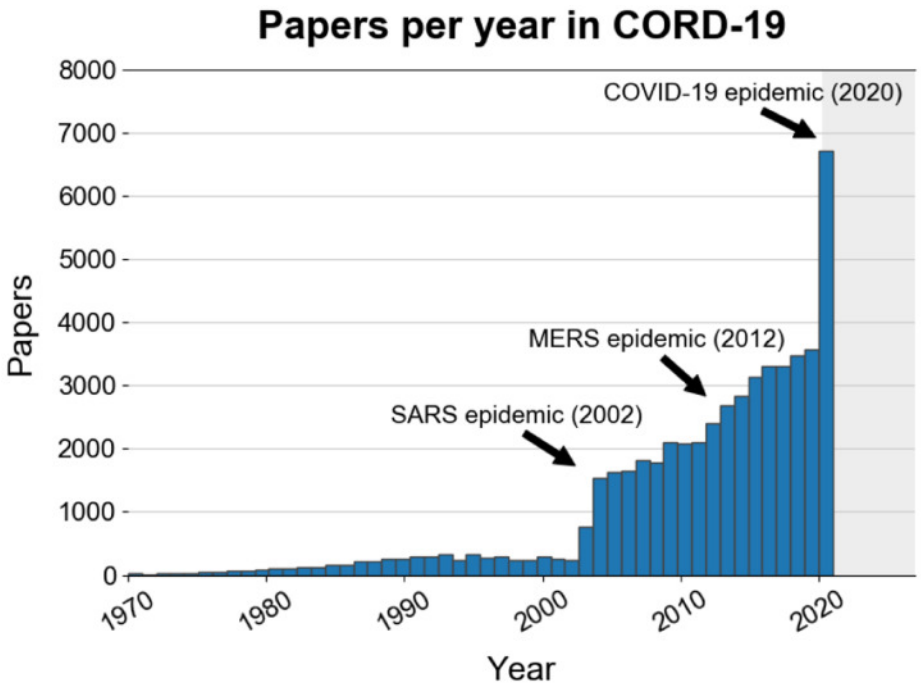


Figura 3.3: Distribuzione annuale dei paper presenti in CORD-19

L'adozione di CORD-19 come risorsa per il data mining è stata accelerata dall'organizzazione di molteplici competizioni legate al Covid-19. Per questo progetto abbiamo scelto di seguire quella organizzata da TREC.

3.2 TREC-COVID Challenge

La *TREC-COVID Challenge* [29] è stata organizzata in collaborazione tra *the Allen Institute for AI*, *the National Institute of Standards and Technology* (NIST), *the National Library of Medicine* (NLM), *Oregon Health and Science University* (OHSU) e *the University of Texas Health Science Center at Houston* (UTHealth). Il task oggetto della competizione riguarda la creazione di sistemi capaci di classificare i documenti presenti in CORD-19 sulla base della loro rilevanza rispetto a determinate query, arrivando a creare un vero e proprio motore di ricerca specializzato sul Covid-19. In particolare, sono previsiti dalla challenge 30 diversi argomenti, incluse query come *How does the coronavirus respond to changes in the weather?* o *What drugs have been active against SARS-CoV or SARS-CoV-2 in animal studies?* Questi argomenti provengono da diverse ricerche e riflettono le domande effettivamente poste dalla comunità internazionale in merito al Covid-19. Dopo la sottomissione dei vari sistemi di IR partecipanti la competizione, esperti di dominio medico hanno avuto

il compito di fornire classifiche e giudizi di pertinenza sul pool di documenti restituiti per ogni query. L'obiettivo di TREC-COVID è quello di fornire giudizi complessivi sulle prestazioni di questi sistemi al fine di focalizzare l'attenzione e gli sforzi della comunità di ricerca internazionale verso le tecniche di information retrieval più efficienti. La competizione si è svolta durante la primavera-estate del 2020 ed ha contato 5 differenti round differenziati in base alle categorie di sistemi accettati.

3.2.1 CO-Search

CO-Search [30] è un *retriever-ranker semantic search engine*, ovvero un motore di ricerca che prende in ingresso delle query, comprese domande in linguaggio naturale, e recupera articoli scientifici basati sulla letteratura relativa al nuovo coronavirus. Esplora i contenuti di più di 128,000 documenti pubblicati sul Covid-19 e appartenenti al *Covid-19 Open Research Dataset*.

Il processo di retrieval viene effettuato utilizzando un *semantic model* e due *keyword model*. Per quanto riguarda il primo, gli autori hanno deciso di creare un grafo bipartito dei paragrafi e degli articoli in essi citati, generando così più di 2.2 milioni di tuple di tipo *(paragraph, title)*. Queste tuple vengono poi utilizzate per addestrare un modello di *Siamese-BERT* (SBERT) [31] sul task binario di classificazione di un titolo come citato da un paragrafo. SBERT viene utilizzato per incorporare query e documenti nello stesso spazio latente, consentendo così di recuperare il *nearest-neighbor* semantico. Successivamente, questi embeddings sono combinati con TF-IDF e BM25. In particolare si effettua una combinazione lineare tra i *paragraph-level retrieval scores* di SBERT e i *document-level retrieval scores* di TF-IDF per generare una lista di documenti e utilizzare la *reciprocal ranked fusion* per combinare questa lista con quella risultata dal retrieval ottenuto da BM25 utilizzando Anserini [32].

Nella fase di ranking si prendono in ingresso gli score del processo di retrieval in due diversi moduli, un modulo di *question answering* e uno di *abstractive summarizer*. Viene quindi addestrato un *multi-hop question answering model* con Wikipedia, che tratta le query come se fossero domande e genera delle risposte dai documenti recuperati. Si addestra poi l'*abstractive summarizer*, composto da un encoder BERT e da un decoder GPT-2 modificato, nello stesso modo per generare un *summary*. Il ranker prende quindi in ingresso i punteggi dei documenti recuperati e li ordina in base al grado in cui i documenti contengono le risposte generate e i summary, restituendo quindi un set di documenti ordinati.

CO-Search è stato valutato sui primi due round della TREC-COVID Challenge, ottenendo il punteggio migliore su tutte le metriche e divenendo di fatto lo stato dell'arte per questa competizione.

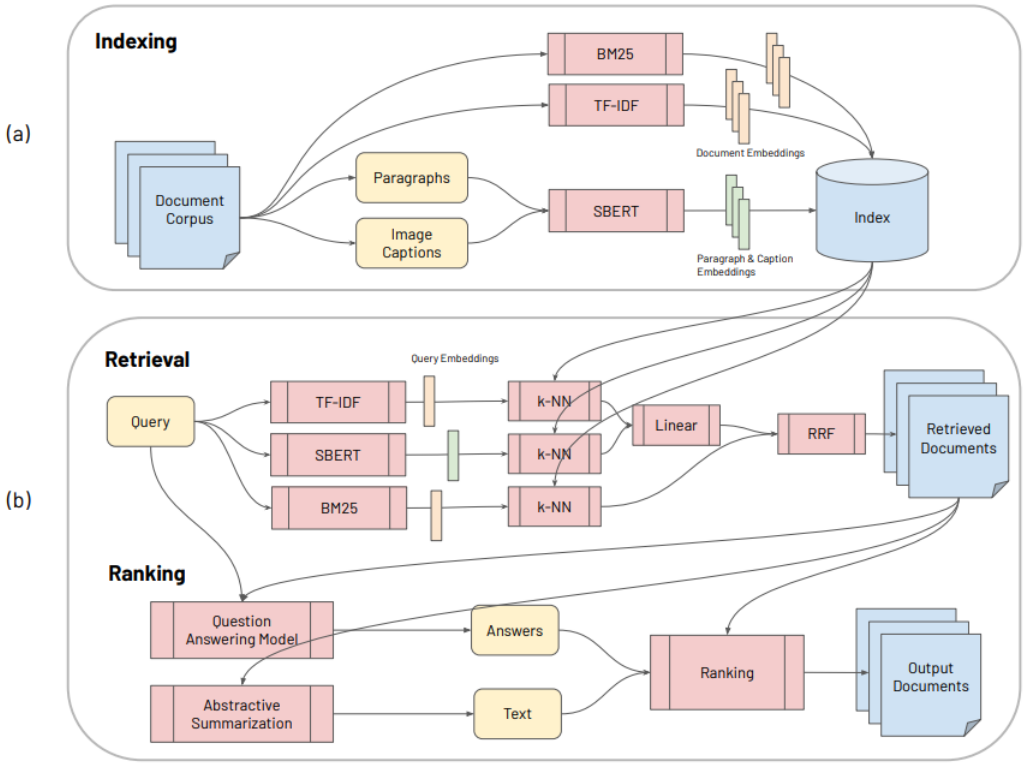


Figura 3.4: Rappresentazione grafica dell'architettura del motore di ricerca CO-Search

	Score	Rank	Score	Rank	Judged@n
Round 1	Automated Systems (102) Judged Pairs (8,691)		All Systems (144) All Pairs (1.53M)		
Bpref	0.5176	1	0.5176	3	-
MAP	0.4870	1	0.2401	21	-
P@5	0.8267	1	0.6333	17	72.00%
P@10	0.7933	1	0.5567	21	65.67%
nDCG@10	0.7233	1	0.5445	21	65.67%
Round 2	Automated Systems (73) Judged Pairs (12,037)		All Systems (136) All Pairs (2.2M)		
Bpref	0.5232	1	0.5402	2	-
MAP	0.5138	1	0.3487	1	-
P@5	0.8171	1	0.8000	3	98.29%
P@10	0.7629	1	0.7200	3	93.71%
nDCG@10	0.7247	1	0.6996	1	93.71%

Figura 3.5: Score ottenuti da CO-Search nei 2 round della TREC-COVID Challenge in cui è risultato vincitore

3.3 Ambiente di lavoro

Per lo sviluppo di questo progetto sono stati utilizzati il linguaggio di programmazione *Python* e il servizio *Google Colaboratory*. Quest'ultimo è un servizio cloud creato da *Google* per sviluppare modelli di machine learning su ambienti virtuali. I modelli utilizzati per il lavoro sono molto complessi e necessitano di risorse adeguate per essere addestrati correttamente, l'utilizzo di un servizio esterno anziché del proprio PC è quindi risultato fondamentale.

3.3.1 Python e Frameworks

Il linguaggio di programmazione scelto per il progetto è la versione 3 del linguaggio *Python*. Si tratta di uno dei linguaggi più comuni in ambito Data Science, in quanto facile da utilizzare e con grandi potenzialità. Esistono inoltre moltissimi frameworks creati dalla comunità che permettono di agevolare ulteriormente il lavoro. Mostriamo quindi di seguito un elenco dei principali frameworks utilizzati per questo progetto:

- **Pickle** - libreria Python che contiene le funzioni per scrivere o leggere oggetti sopra o da un file;
- **NumPy** - libreria Python che aggiunge supporto a grandi matrici e array multidimensionali, insieme a una vasta collezione di funzioni matematiche di alto livello per poter operare efficientemente su queste strutture dati;
- **Pandas** - libreria Python per la manipolazione e l'analisi dei dati. In particolare, offre strutture dati e operazioni per manipolare tabelle numeriche e serie temporali;
- **Natural Language Toolkit (NLTK)** [33] - suite di librerie per l'analisi simbolica e statistica nel campo dell'elaborazione del linguaggio naturale;
- **Rank-BM25** - libreria contenente algoritmi per ricercare all'interno di una collezione di documenti quelli più rilevanti per una determinata query (e.g. Okapi BM25);
- **Scikit-learn** [34] - libreria open source di machine learning per il linguaggio di programmazione Python;
- **Pytorch** [35] - libreria software open source per il machine learning a sua volta basata sulla libreria Torch, utilizzata principalmente per visione artificiale e Natural Language Processing;

- **HuggingFace** - libreria software open source che consente di creare, addestrare e distribuire modelli all'avanguardia basati sullo stato dell'arte relativo al Natural Language Processing;
- **Transformers** [36] - libreria di HuggingFace che garantisce un rapido accesso ai modelli di Transformers che costituiscono lo stato dell'arte di NLP (e.g. BERT, SciBERT, ecc.);
- **PyTorch Metric Learning** [37] - libreria basata su Pytorch che fornisce funzioni utili per facilitare l'implementazione di algoritmi basati sul Deep Metric Learning.

3.3.2 Google Colaboratory

Google Colaboratory o, in breve, **Colab** è un servizio rilasciato da *Google Research*. Permette a chiunque di scrivere ed eseguire codice Python tramite il browser e, per via delle ingenti risorse messe a disposizione, è particolarmente adatto per la Data Science e in particolare per il machine learning. Molti dei più comuni frameworks Python sono già installati, ma è sempre possibile aggiungerne di nuovi attraverso il comando *pip*. Il servizio è basato su **Jupyter Notebook** e consente dunque di aggiungere celle di testo o di codice ed eseguirle. Tutti i notebook vengono eseguiti su macchine virtuali con un grande quantitativo di risorse a disposizione.

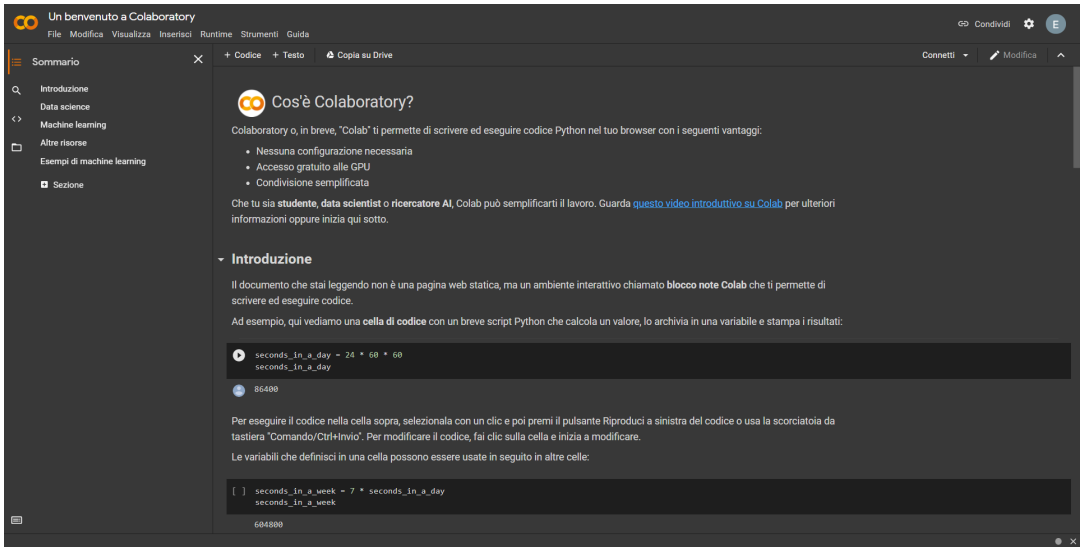


Figura 3.6: Interfaccia di Google Colaboratory

Gli utenti possono inoltre scegliere quale hardware utilizzare tra tre differenti configurazioni di accelerazione:

- **None.** Questa configurazione permette di utilizzare solamente la CPU della macchina virtuale messa a disposizione da Colab.
- **GPU Acceleration.** Questa configurazione permette di utilizzare una macchina virtuale con a disposizione una GPU e Cuda.
- **TPU Acceleration.** Questa configurazione permette di utilizzare una macchina virtuale con a disposizione una TPU.

L'utilizzo di Colab è totalmente gratuito, tuttavia l'accesso a risorse come GPU e TPU prevede dei limiti di utilizzo giornalieri che variano in base alla richiesta e all'impiego (e.g. vengono assegnate risorse per un quantitativo di tempo maggiore agli utenti che fanno un utilizzo interattivo di colab piuttosto che a quelli che eseguono un lungo addestramento). Con l'abbonamento a Colab Pro è possibile ottenere un accesso più affidabile a risorse migliori. Non è però possibile scegliere il modello e.g. di GPU che verrà assegnato, in quanto questi variano nel tempo e a seconda delle disponibilità. Solitamente si ha l'accesso a GPU come Nvidia K80, Nvidia Tesla T4, P4 e P100.

Capitolo 4

Esperimenti e risultati

Durante lo sviluppo di questo progetto sono stati condotti svariati test ed esperimenti, alcuni hanno fallito mentre altri hanno ottenuto risultati apprezzabili. In questo capitolo analizzeremo tutti i passaggi che sono stati necessari al fine di addestrare diversi modelli e testarli su un sistema di information retieval completo attraverso il quale ne abbiamo valutato le potenzialità. Confronteremo in seguito i risultati ottenuti sulla TREC-COVID Challenge con quelli performati da CO-Search, che ne costituisce di fatto lo stato dell'arte.

4.1 Implementazione

Il primo passo per la creazione di un motore di ricerca è sicuramente quello di riuscire ad addestrare uno o più modelli in grado di accostare tra loro efficacemente frasi o insiemi di parole semanticamente simili. Per prima cosa abbiamo provveduto a scaricare e ad analizzare il dataset completo della competizione TREC-COVID Challenge, il quale era disponibile in formato json. Si è quindi reso necessario implementare alcune funzioni per estrarre gli elementi di interesse dal file scaricato, ovvero titolo, abstract e testo completo.

```
In [1]:
def get_document_title(document_id):
  document = "/content/pdf_json/" + document_id + ".json"

  with open(document) as data_file:
    data = json.load(data_file)

  title = data["metadata"]["title"]

  return title
```

```
In [2]:
def get_document_abstract(document_id):
  document = "/content/pdf_json/" + document_id + ".json"

  with open(document) as data_file:
    data = json.load(data_file)

  full_abstract = ""
  for index in range (len(data["abstract"])):
    full_abstract += data["abstract"][index]["text"]
    full_abstract += " "

  return full_abstract

In [3]:
def get_document_full_text(document_id):
  document = "/content/pdf_json/" + document_id + ".json"

  with open(document) as data_file:
    data = json.load(data_file)

  paragraphs = [get_document_title(document_id),
  get_document_abstract(document_id)]
  for index in range (len(data["body_text"])):
    paragraphs.append(data["body_text"][index]["text"])

  return paragraphs
```

Come è possibile notare dalla terza funzione, osserviamo che abbiamo deciso di aggiungere al testo completo, sotto la forma dei primi due paragrafi, anche il titolo e l'abstract di ogni documento in analisi.

Successivamente si è quindi deciso di sfruttare queste funzioni per salvare i dati del dataset all'interno di un DataFrame pandas, in modo da renderli più leggibili e facilmente analizzabili. In particolare, le colonne di questo DataFrame contengono rispettivamente per ogni documento titolo, elenco dei paragrafi sotto forma di lista e numero di paragrafi costituenti il documento.

```
In [4]:
import os
counter = 0
papers = pd.DataFrame(columns=["id", "paragraphs",
"paragraphs_number"])
```

```
directory = "/content/pdf_json"
for filename in os.listdir(directory):
  filename_no_extension = filename.rsplit('.', 1)[0]
  paragraphs = get_document_full_text(filename_no_extension)
  papers.loc[counter] = filename_no_extension, paragraphs,
  len(paragraphs)
  counter += 1
```

Dall'analisi del dataset è risultata la presenza di 40144 documenti contenenti mediamente 37 paragrafi, compresi il titolo e l'abstract. La prima criticità emersa riguardava le possibilità di addestramento di un modello su una così grande mole di dati. Infatti, dopo alcune prove e considerazioni, è emerso che con la memoria grafica dedicata messa a disposizione dalle GPU di Google Colaboratory (solitamente tra i 12 e i 17GB) non fosse possibile addestrare modelli utilizzando documenti che fossero più lunghi di 5 paragrafi. Sulla base di queste premesse abbiamo quindi inizialmente deciso di considerare solo quei documenti che avessero un numero di paragrafi prossimo al 5, per evitare di perdere contenuto informativo in caso di taglio di documenti più lunghi. Per i primi test ci è sembrato sufficiente considerare 600 documenti e, per poter raggiungere tale numero, abbiamo considerato tutti i paper del dataset costituiti da 5 e da 6 paragrafi (questi ultimi privati dell'ultimo paragrafo).

```
In [5]:
filtered_papers = papers[(papers.paragraphs_number <= 6) &
(papers.paragraphs_number >= 5)]
for it in filtered_papers.index:
  filtered_papers["paragraphs"][it] =
  filtered_papers["paragraphs"][it][:5]
```

La seconda criticità emersa, come ampiamente spiegato in fase di introduzione di questa tesi, ha riguardato il dataset, totalmente non supervisionato. I vari paper sono quindi in linguaggio naturale e non vi è alcuna label che possa aiutare ad individuarne le caratteristiche. Con questi dati era quindi impossibile addestrare qualsiasi tipo di modello basato su reti neurali, in quanto esse richiedono obbligatoriamente per funzionare una X (dati in input) e una y (output da determinare, label). Per risolvere il problema abbiamo deciso di adottare una soluzione di tipo **self-supervised** dove il modello è addestrato a posizionare, in uno spazio latente, parti di testo semanticamente simili vicine e allontanare quelle che trattano argomenti diversi. Per fare ciò, abbiamo considerato come esempi simili i paragrafi provenienti dallo stesso documento. È logico pensare che se due paragrafi appartengono allo stesso documento, essi debbano essere correlati a livello semantico. Al contrario, abbiamo preso

quelli provenienti da documenti diversi come dissimili, quindi da allontanare. Si considerano quindi come X (trainset) le tuple formate dai vari paragrafi dei 600 documenti e come y i relativi id del documento di origine.

4.1.1 ScientificBERT model e Set-up

Il modello scelto per questo progetto è ScientificBERT (o SciBERT), in quanto pre-addestrato appositamente su documenti a carattere scientifico, come quelli che costituiscono il nostro dataset. Ha inoltre raggiunto lo stato dell'arte su diversi task di NLP, confermandosi uno dei migliori modelli di transformers basati su BERT. Un'altra possibile scelta sarebbe potuta essere GPT-3, ma il pre-addestramento specifico di SciBERT lo ha reso maggiormente adatto a questo task. ScientificBERT è disponibile in due diverse versioni:

- **scibert-scivocab**. Modello SciBERT che utilizza un vocabolario scientifico, disponibile sia nella versione uncased che in quella cased;
- **scibert-basevocab**. Modello SciBERT che utilizza il vocabolario standard di BERT base, disponibile sia nella versione uncased che in quella cased.

Per questo progetto si è deciso di utilizzare *scibert-scivocab-uncased*, in quanto è la versione che ha ottenuto i risultati più performanti su tutti i principali task di NLP a carattere scientifico.

Ci sono molti modi per ottenere un modello di SciBERT ed è anche possibile svilupparlo per conto proprio utilizzando *TensorFlow* [38] o *Pytorch*, tuttavia questo approccio può essere molto complesso e dispendioso in termini di tempo. Per il progetto è stata utilizzata la libreria **Trasnformers** di *HuggingFace*, grazie alla quale è possibile ottenere con poche righe di codice un modello si SciBERT funzionante da poter modificare. Su Google Colaboratory è molto semplice installare questa libreria, è infatti sufficiente digitare:

```
In [1]: pip install transformers
```

Dopo la scelta del modello è stato quindi possibile procedere con il download del tokenizer da utilizzare nella fase di pre-processing dei dati.

```
In [2]:
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained(
            "allenai/scibert_scivocab_uncased")
```

Successivamente si è quindi utilizzato il tokenizer per ottenere gli *input_ ids* e le *attention_ mask* necessarie come input al modello.

```
In [3]:
input_ids = [tokenizer(par, add_special_tokens=True,
             max_length=512, padding="max_length",
             return_tensors="pt", return_token_type_ids=False,
             truncation=True) for par in
             papers_dataframe["paragraphs"].values.tolist()]
papers_dataframe["input_ids"] = [x["input_ids"] for x in input_ids]
papers_dataframe["attention_mask"] = [x["attention_mask"]
                                      for x in input_ids]
```

Abbiamo infine ricreato il dataset nel formato di *Pytorch*, in modo da renderlo versatile e compatibile con il modello di SciBERT della libreria Transformers, essendo essa basata su *Pytorch*.

4.1.2 Applicazione delle tecniche di Deep Metric Learning

Il **deep metric learning**, come già ampiamente spiegato nel corso di questo documento, è nato per la computer vision e utilizza le reti neurali profonde per individuare feature discriminanti a partire dalle immagini, apprendendo così come distinguere immagini che rappresentano un oggetto da quelle che ne rappresentano un altro. Dato che sia le immagini che il testo vengono rappresentati allo stesso modo nello spazio di embeddings, ovvero sotto forma di vettori di numeri reali, abbiamo pensato di provare ad adattare queste tecniche anche all'elaborazione del linguaggio naturale e in particolare al task oggetto di questo progetto. La supposizione è stata che se le tecniche di deep metric learning sono in grado di individuare immagini tra loro simili e rappresentarle in posizioni tra loro vicine si può fare lo stesso anche con porzioni di testo aventi lo stesso significato semantico. Abbiamo dunque deciso di riadattare un sistema di questo genere sostituendo la rete neurale convoluzionale, generalmente utilizzata per task di visione artificiale, con il modello di SciBERT precedentemente illustrato.

Successivamente è stato necessario utilizzare *miner* e *loss* tipici del deep metric learning, sfruttando quindi il primo per filtrare le relazioni più interessanti da considerare per le varie loss che abbiamo deciso di testare. Per effettuarne l'implementazione si è scelto di usufruire della libreria *PyTorch Metric Learning* che, essendo anch'essa basata su PyTorch è perfettamente compatibile con Trasnformers di HuggingFace. Questo framework permette di richiamare

attraverso comode funzioni le loss e i miner più comuni del deep metric learning. Anche in questo caso l'installazione su Google Colaboratory è piuttosto semplice, in quanto è sufficiente digitare:

```
In [1]: pip install pytorch-metric-learning
```

seguito da

```
In [2]: from pytorch_metric_learning import losses, miners
```

per effettuare l'import dei moduli di nostro interesse.

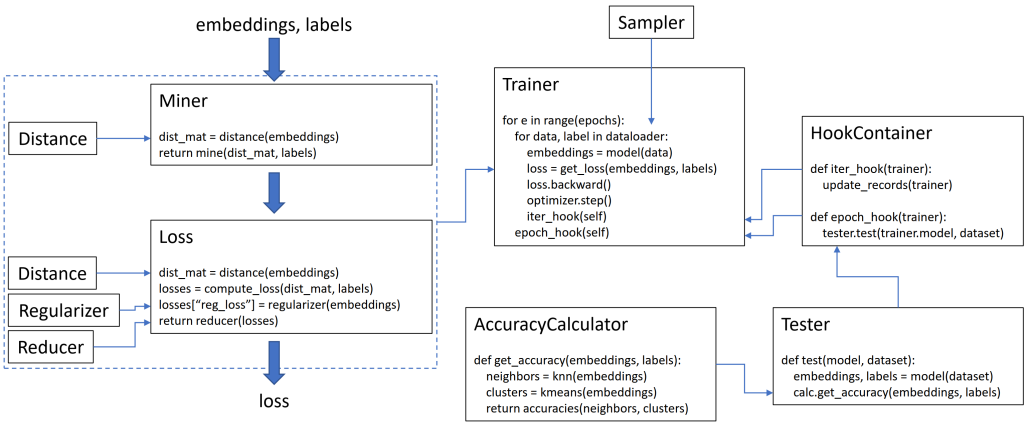


Figura 4.1: Schema delle funzionalità in ambito deep metric learning implementabili con l'ausilio del frameword PyTorch Metric Learning

Procediamo ora con una rapida spiegazione degli iperparametri delle loss utilizzate, il cui funzionamento è già stato illustrato in maniera dettagliata nel capitolo relativo alle *Tecnologie Disponibili* e in particolare nella sezione 2.11.

Triplet Loss

La triplet loss è la funzione più utilizzata in ambito deep metric learning e riguarda la suddivisione del mini-batch considerato in triplette costituite da un'ancora, un elemento con label simile all'ancora e un elemento con label differente. Gli iperparametri che è possibile impostare sono:

- **margin**: la differenza desiderata tra la distanza ancora-elemento positivo e quella tra ancora-elemento negativo. Si tratta del valore che abbiamo definito α in fase di spiegazione di questa loss nella sottosezione 2.11.2;

- **swap**: iperparametro booleano che, in caso di valore *True*, utilizza la distanza elemento positivo-elemento negativo anziché quella ancora-elemento negativo, se questa viola maggiormente il margine;
- **smooth_loss**: iperparametro booleano che, in caso di valore *True*, utilizza la versione $_{\text{log-exp}}$ della triplet loss;
- **triplets_per_anchor**: iperparametro che indica il numero di triplette per elemento da campionare all'interno di un mini-batch. Come valori può assumere un numero intero o la stringa *all* (default) che indica di considerare tutte le combinazioni possibili.

Accoppiato a questa loss si è scelto di utilizzare il *Triplet Miner*, che possiede due diversi iperparametri:

- **margin**: ha significato identico al margine della Triplet Loss e devono essere impostati uguali;
- **type_of_triplets**:
  - *all*: considera tutte le triplette che superano il margine
  - *hard*: è una sotto-categoria di *all* in cui il negativo è più vicino all'ancora del positivo
  - *semihard*: è una sotto-categoria di *all* in cui il negativo è più lontano dall'ancora del positivo
  - *easy*: considera tutte le triplette che non violano il margine

Osserviamo ora un esempio di codice per il training di un modello utilizzando *SciBERT*, la *triplet-margin-loss* e il *triplet-margin-miner*.

```
In [1]:
device = torch.device("cuda")

model = AutoModel.from_pretrained("allenai/scibert_scivocab_uncased")
                 .to(device)
optimizer = optim.Adam(model.parameters(), lr=0.00001)
num_epochs = 5

loss_func = losses.TripletMarginLoss(margin=0.2)
mining_func = miners.TripletMarginMiner(margin=0.2)

for epoch in range(1, num_epochs + 1):
  model.train()
```

```
for batch_idx, batch in tqdm(enumerate(train_loader)):
  data, mask, labels = torch.reshape(batch["input_ids"],
                       (10, 512)).to(device),
                       torch.reshape(batch["attention_mask"],
                       (10, 512)).to(device),
                       torch.reshape(batch["label"],
                       (10, ))
  optimizer.zero_grad()
  output = model(data, mask)
  embeddings = torch.mean(output[0], dim=1)
  indices_tuple = mining_func(embeddings, labels)
  loss = loss_func(embeddings, labels, indices_tuple)
  loss.backward()
  optimizer.step()
  if batch_idx % 20 == 0:
    print("Epoch  Iteration : Loss = ,
          Number of mined triplets = "
          .format(epoch, batch_idx, loss, mining_func.num_triplets))
  del loss, embeddings, output
```

Altri importanti iperparametri da considerare per l'addestramento sono:

- **Numero di epoche**: numero di volte in cui il modello viene addestrato sull'intero dataset, in questo caso si è scelto di utilizzarne 5;
- **Optimizer**: l'ottimizzatore, in questo caso *Adam*, che guida il modello durante il processo di addestramento e attraverso il quale è possibile impostare il *learning rate* (e.g. 0.00001).

Angular Loss

Anche l'angular loss funziona con il meccanismo delle triplette formate da ancora, elemento positivo ed elemento negativo. Tuttavia fa uso della distanza angolare tra i vari elementi anziché di quella euclidea, ovvero si utilizza il coseno degli angoli del triangolo generato dal collegamento dei tre elementi di ciascuna tripletta per determinarne la correlazione.

Nel caso di questa loss è possibile impostare un solo iperparametro:

- **alpha**: l'angolo minimo da considerare specificato in gradi. Nel paper vengono utilizzati valori compresi tra 36 e 55.

Anche in supporto all'*angular-loss* è previsto l'angular-miner, avente anch'esso un solo iperparametro da impostare:

- **angle**: il miner restituisce solamente le triplette che hanno un angolo maggiore di quello impostato da questo iperparametro.

Lifted Structure Loss

La Lifted Structure Loss funziona in maniera simile alla Triplet Loss, ma, anziché considerare solamente le due distanze anchor-positive e anchor-negative per ogni ancora, utilizza tutte le possibili distanze tra le coppie di elementi *anchor-positive* presenti nel mini-batch considerato. Questo procedimento riduce la velocità di convergenza dell'algoritmo e per questo spesso si utilizzano approcci *mining-based* per considerare solamente un numero ridotto di cosidetti elementi *hard-negatives* per ogni coppia *anchor-positive*.

Gli iperparametri che caratterizzano questa loss sono:

- **pos_margin**: margine degli elementi positivi, nel paper impostato a 0;
- **neg_margin**: margine degli elementi negativi, nel paper impostato a 1.

In questo caso si è deciso di utilizzare il *triplet-margin-miner* già precedentemente illustrato.

Multi Similarity Loss

La Multi Similarity Loss è una particolare loss dell'ambito deep metric learning molto recente che si basa sull'utilizzo combinato di tre differenti similarità: la *Self Similarity*, la *Negative Relative Similarity* e la *Positive Relative Similarity*. Si suggerisce inoltre di utilizzare per l'addestramento solo *hard negative pairs* e *hard positive pairs* filtrabili attraverso tecniche di tipo *mining-based*.

Gli iperparametri che caratterizzano questa loss sono:

- **alpha**: rappresenta il peso applicato alle coppie di elementi positivi, 2 secondo il paper;
- **beta**: rappresenta il peso applicato alle coppie di elementi negativi, 50 secondo il paper;
- **base**: rappresenta l'offset applicato all'esponente della loss, 1 secondo il paper. Nella formula è indicato da λ.

Il mining dei cosiddetti *hard pairs* viene effettuato attraverso il *multi-similarity-miner*, impostabile attraverso il seguente iperparametro:

- **epsilon**:

– i *negative pairs* sono scelti se hanno una similarità maggiore rispetto all'*hardest positive pair* dopo avergli sottratto *epsilon*;

– i *positive pairs* sono scelti se hanno una similarità minore rispetto all'*hardest negative pair* dopo avergli sommato *epsilon*.

4.1.3 Primi Test

Ecco di seguito le prime configurazioni di modelli addestrati usufruendo delle loss sopraelencate:

Tabella 4.1: Prime configurazioni addestrate per la Triplet Loss

num	**n-doc**	**modello**	**lr**	**miner**	**m-mrg**	**loss**	**l-mrg**
1	280	SciBERT	0.000001	TMM	0.2	TML	0.2
2	600	SciBERT	0.000001	TMM	0.2	TML	0.2
3	600	SciBERT	0.00005	TMM	0.2	TML	0.2
4	600	SciBERT	0.00001	TMM	0.2	TML	0.2
5	600	SciBERT	0.000005	TMM	0.2	TML	0.2
6	600	SciBERT	0.00001	TMM	0.1	TML	0.1
7	600	SciBERT	0.000005	TMM	0.1	TML	0.1
8	600	SciBERT	0.000001	TMM	0.1	TML	0.1
9	600	SciBERT	0.00005	TMM	0.1	TML	0.1
10	600	SciBERT	0.00001	TMM	0.3	TML	0.3
11	600	SciBERT	0.00005	TMM	0.3	TML	0.3
12	600	SciBERT	0.000005	TMM	0.3	TML	0.3
13	600	SciBERT	0.000001	TMM	0.3	TML	0.3
14	600	SciBERT	0.000005	TMM	0.4	TML	0.4
15	600	SciBERT	0.000001	TMM	0.4	TML	0.4
16	600	SciBERT	0.00005	TMM	0.4	TML	0.4
17	600	SciBERT	0.00001	TMM	0.4	TML	0.4

Tabella 4.2: Prime configurazioni addestrate per Angular Loss

num	**n-doc**	**modello**	**lr**	**miner**	**m-ang**	**loss**	**l-α**
18	600	SciBERT	0.00001	AM	20	AL	20
19	600	SciBERT	0.00001	AM	40	AL	40
20	600	SciBERT	0.00001	AM	50	AL	50
21	600	SciBERT	0.00001	AM	20	AL	70
22	600	SciBERT	0.00001	AM	30	AL	40
23	600	SciBERT	0.00001	AM	5	AL	5
24	600	SciBERT	0.000001	AM	5	AL	5

Tabella 4.3: Prime configurazioni addestrate per Lifted Structure Loss

num	**n-doc**	**modello**	**lr**	**miner**	**m-mrg**	**loss**	**n-m**	**p-m**
25	600	SciBERT	0.00001	TMM	0.1	LSL	1	0
26	600	SciBERT	0.00001	TMM	0.01	LSL	1	0

Tabella 4.4: Prime configurazioni addestrate per Multi Similarity Loss

num	**n-doc**	**modello**	**lr**	**miner**	**epsilon**	**loss**	**l-α**	**l-β**	**l-b**
27	600	SciBERT	0.00001	MSM	0.01	MSL	2	50	0.5
28	600	SciBERT	0.00001	MSM	0.0001	MSL	2	50	0.5
20	600	SciBERT	0.00001	MSM	0.001	MSL	2	50	0.5
30	600	SciBERT	0.00001	MSM	0.00001	MSL	2	50	0.5
31	600	SciBERT	0.00001	MSM	0.0001	MSL	4	100	1
32	600	SciBERT	0.00001	MSM	0.0001	MSL	1	25	0.25
33	600	SciBERT	0.00001	MSM	0.0001	MSL	3	150	0.75
34	600	SciBERT	0.00001	MSM	0.0001	MSL	8	200	2
35	600	SciBERT	0.00001	MSM	0.0001	MSL	2	100	0.5
36	600	SciBERT	0.00001	MSM	0.0001	MSL	3	100	1
37	600	SciBERT	0.00001	MSM	0.0001	MSL	4	200	2
38	600	SciBERT	0.000005	MSM	0.0001	MSL	5	150	0.75
39	600	SciBERT	0.000001	MSM	0.0001	MSL	5	150	0.75
40	600	SciBERT	0.00005	MSM	0.0001	MSL	5	150	0.75

Nelle configurazioni sopraelencate bisogna presupporre che di tutti i documenti sono stai considerati solamente 5 paragrafi per questioni di risorse hardware disponibili e, in particolare, i documenti presi in esame ne avevano 5 o 6 (quelli a 6 paragrafi sono stati troncati). Tutti gli addestramenti sono stati

eseguiti per 5 epoche con un batch-size di 10 paragrafi, ovvero 2 documenti. Inoltre, le sigle presenti nelle voci miner e loss delle singole tabelle hanno i seguenti significati:

- TMM: Triplet Margin Miner
- TML: Triplet Margin Loss
- AM: Angular Miner
- AL: Angular Loss
- LSL: Lifted Structure Loss
- MSM: Multi Similarity Miner
- MSL: Multi Similarity Loss

4.2 Valutazione dei modelli

Per effettuare la valutazione dei modelli addestrati abbiamo scelto di utilizzare un sistema di *information retrieval*, in modo tale da poter poi confrontare i risultati ottenuti con quelli di *CO-Search*.

4.2.1 Il sistema di IR

I migliori modelli neurali della TREC-COVID Challenge sono stai inseriti all'interno di un sistema di *information retrieval* per testarne performance e capacità, così anche noi abbiamo deciso di procedere in una direzione analoga. In particolare, il sistema di IR che abbiamo utilizzato è formato da tre differenti parti:

- **Indexer:** crea le strutture dati per l'indicizzazione dei documenti. Nello specifico un modello neurale crea gli embeddings e BM25-Okapi effettua la vettorizzazione TF-IDF.
- **Retriever:** prende in ingresso una query in linguaggio naturale (e.g. frase che si inserisce su un normale motore di ricera) e restituisce i documenti più pertinenti ad essa analizzandone solo titolo ed abstract.
- **Reranker:** controlla il contenuto completo dei primi 10 documenti restituiti dal *retriever*, creando un vettore per ogni paragrafo di ognuno di essi. Cerca poi i vettori più pertinenti alla query riordinando la lista dei 10 migliori documenti secondo le nuove informazioni acquisite.

4.2.2 Le metriche utilizzate

Per la valutazione dei modelli inseriti all'interno del sistema di IR abbiamo utilizzato le stesse metriche della TREC-COVID Challenge, in particolare queste sono:

- **Precision (P):**

$$P@N = \frac{|documenti\ rilevanti\ tra\ i\ primi\ N|}{N}$$

  Nello specifico abbiamo utilizzato P@5 e *P@10*.

- **Normalized discounted cumulative gain (nDCG):** Per la posizione $i \in \{0, 1, ..., N\}$, il *normalized discounted cumulative gain* del set di documenti recuperati in Q query è dato da:

$$nDCG@N = \frac{1}{Q}\sum_{q=1}^{Q}\frac{DCG_{\mathrm{p}}{}^{(\mathrm{q})}}{IDCG_{\mathrm{p}}{}^{(\mathrm{q})}},\ conDCG_{\mathrm{p}}{}^{(\mathrm{q})} = rel_1{}^{(\mathrm{q})} + \sum_{i=2}^{N}\frac{rel_{\mathrm{i}}{}^{(\mathrm{q})}}{log_2(i)}$$

  Dove $rel_{\mathrm{i}}{}^{(\mathrm{q})}$ indica la rilevanza della entry i, ordinata secondo la pertinenza con la query *q*. IDCG indica invece il valore ideale, ovvero il massimo assumibile da DCG. *nDCG* è molto utile per valutare le performance dei motori di ricerca e per questo lavoro abbiamo utilizzato *nDCG@10*.

- **Mean average precision (MAP):**

$$MAP = \frac{1}{Q}\sum_{q=1}^{Q}\int_0^1 P_{\mathrm{q}}(R)dR$$

  Dove R è la recall, mentre P_q è la precisione in funzione della recall per una determinata query.

- **Binary preference (Bpref):**

$$Bpref = \frac{1}{R}\sum_{r=1}^{R}1 - \frac{|n\ valutati\ meglio\ di\ r|}{R}$$

  Dove R è il numero di documenti giudicati rilevanti, r è uno dei documenti giudicati rilevanti e n è uno dei primi R documenti irrilevanti recuperati.

4.2.3 I risultati ottenuti

Prima di ottimizzare le operazioni di *retrieval* e *reranking* abbiamo deciso di testare tutti i modelli addestrati su un *retrieval* con iperparametro α casuale, in modo da selezionare solamente quelli più promettenti. Il test è avvenuto per ogni configurazione su ognuna delle 5 epoche, ma riportiamo per semplicità nella tabella 4.5 solamente quella che ha ottenuto le performance migliori (che è comunque la quinta epoca in tutti i casi, non si sono verificate dunque situazioni di overfitting). Per la valutazione sono state utilizzate tutte le metriche definite nella sezione precedente.

I cinque modelli scelti per procedere con la valutazione sono stati il *4*, addestrato con *Triplet Loss*, il *27*, il *28*, il *30* e il *35* addestrati invece con *Multi Similarity Loss*. Come prevedibile, la *MSL*, in quanto più recente e di fatto costituente lo stato dell'arte del *deep metric learning* è stata la più performante, seguita dalla *Triplet Loss*. *Angular Loss* e *Lifted Structure Loss* non hanno invece ottenuto risultati particolarmente rilevanti e significativi, in quanto probabilmente meno adatte ad essere riconvertite per l'utilizzo in un task di questo tipo. Una volta ottenuti i modelli migliori, abbiamo deciso di provare a ritestarne l'addestramento variando il numero dei documenti utilizzati. Purtroppo, per via delle limitate risorse a disposizione (memoria grafica dedicata delle GPU di Google Colaboratory), non è stato possibile aumentare il numero dei paragrafi esaminati per l'addestramento elevandolo al di sopra del 5. Si è scelto di effettuare due differenti tentativi, il primo ha comportato una riduzione dei documenti considerati per l'addestramento a 280, velocizzando di fatto il tempo di training. In questo caso sono stati utilizzati solamente paper che contavano effettivamente 5 paragrafi ma, come prevedibile, i risultati ottenuti sono stati più scarsi. Il secondo tentativo ha comportato l'aumento dei documenti considerati alla cifra più alta possibile. Dato che questa scelta comporta un notevole incremento dei tempi di addestramento abbiamo dovuto tenere conto dei limiti di utilizzo delle GPU di Google Colaboratory, che, solitamente, si aggirano intorno alle 4 o 5 ore al giorno. Abbiamo dunque cercato di ponderare la scelta per effettuare sessioni di training che non eccedessero tale durata. Il numero di paper che è stato possibile utilizzare è di 2500, ottenendo un tempo di addestramento dei modelli che si aggirava intorno alle 4 ore variando leggermente a seconda della tipologia di GPU assegnata da Colab in quel determinato momento. Contrariamente alle aspettative i risultati non sono migliorati e, anzi, in alcuni casi sono stati leggermente peggiori rispetto all'utilizzo dei 600 documenti, mantenendosi comunque in generale sulla stessa linea. Ci siamo spiegati questo risultato con il fatto che, per utilizzare un così alto numero di documenti in fase di addestramento, si è reso necessario considerare anche quelli con un numero di paragrafi superiore al 10, che sono

Tabella 4.5: Risultati ottenuti dai modelli addestrati con le prime configurazioni. I cinque modelli più performanti, scelti per le fasi successive, sono contrassegnati in grassetto

num	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
1	0.47	0.39	0.51	0.17	0.40
2	0.47	0.39	0.52	0.18	0.40
3	0.55	0.46	0.52	0.20	0.40
4	**0.60**	**0.50**	**0.56**	**0.22**	**0.44**
5	0.47	0.40	0.50	0.17	0.40
6	0.57	0.50	0.58	0.23	0.45
7	0.52	0.45	0.56	0.21	0.44
8	0.51	0.46	0.56	0.21	0.43
9	0.52	0.50	0.54	0.22	0.41
10	0.47	0.41	0.49	0.18	0.39
11	0.47	0.38	0.45	0.16	0.36
12	0.49	0.44	0.50	0.18	0.40
13	0.44	0.38	0.49	0.17	0.39
14	0.47	0.42	0.48	0.17	0.38
15	0.42	0.39	0.47	0.15	0.37
16	0.43	0.36	0.41	0.13	0.34
17	0.41	0.36	0.41	0.12	0.34
18	0.46	0.39	0.45	0.17	0.38
19	0.34	0.37	0.52	0.18	0.41
20	0.12	0.09	0.26	0.03	0.21
21	0.44	0.40	0.58	0.22	0.44
22	0.43	0.40	0.56	0.22	0.44
23	0.38	0.32	0.39	0.12	0.32
24	0.41	0.35	0.44	0.14	0.34
25	0.52	0.42	0.54	0.19	0.41
26	0.40	0.34	0.47	0.16	0.35
27	**0.64**	**0.56**	**0.64**	**0.29**	**0.49**
28	**0.61**	**0.56**	**0.64**	**0.29**	**0.50**
29	0.59	0.51	0.62	0.27	0.48
30	**0.61**	**0.57**	**0.64**	**0.29**	**0.49**
31	0.40	0.33	0.47	0.15	0.35
32	0.53	0.48	0.62	0.27	0.48
33	0.55	0.52	0.61	0.26	0.45
34	0.39	0.33	0.46	0.15	0.35
35	**0.63**	**0.53**	**0.63**	**0.30**	**0.49**
36	0.41	0.33	0.47	0.15	0.35
37	0.39	0.33	0.46	0.15	0.35
38	0.55	0.50	0.61	0.25	0.46
39	0.50	0.44	0.58	0.22	0.45
40	0.50	0.44	0.58	0.22	0.45

Tabella 4.6: Test con 280, 600 e 2500 paragrafi per il modello 4

num	***n-doc***	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
4	280	0.41	0.37	0.44	0.13	0.35
4	600	0.60	0.50	0.56	0.22	0.44
4	2500	0.62	0.49	0.54	0.21	0.42

Tabella 4.7: Test con 280, 600 e 2500 paragrafi per il modello 27

num	***n-doc***	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
27	280	0.51	0.49	0.60	0.24	0.47
27	600	0.64	0.56	0.64	0.29	0.49
27	2500	0.60	0.53	0.63	0.28	0.48

stati tutti troncati alla lunghezza di soli 5 per via della limitata memoria grafica dedicata disponibile sulle GPU. Ne è risultato che probabilmente la perdita di contenuto informativo causata dalla troncatura dei paper non ha permesso al modello di apprendere un maggior numero di elementi che potessero guidarlo poi nel processo di valutazione effettuato in fase di test.

Riportiamo nelle tabelle 4.6, 4.7, 4.8, 4.9 e 4.10 i risultati ottenuti per ciascuno dei 5 modelli più performanti con le tre differenti configurazioni, ovvero utilizzando 280, 600 e 2500 documenti. Anche in questo caso si è sempre effettuato un addestramento per 5 epoche e si sono sempre considerati solamente i primi 5 paragrafi di ciascun documento, utilizzando un batch size di 10 paragrafi, i.e. 2 documenti.

Dopo questo secondo ciclo di addestramenti abbiamo comunque deciso di continuare a considerare la versione a 600 paragrafi di ciascuno dei 5 modelli precedentemente scelti e li abbiamo utilizzati per effettuare le operazioni di retrieval e reranking. In particolare, per la prima, abbiamo testato su ogni modello i valori dell'iperparametro α compresi tra 0.4 e 1, con passo 0.05. Questo iperparametro rappresenta nello specifico il valore del contributo di *BM25Okapi*, ovvero in quale misura viene utilizzato questo e in quale misura viene utilizzato il modello *SciBERT* per le operazioni di retrieval ($\alpha = 0$ indica di utilizzare solamente *BM25Okapi*, mentre $\alpha = 1$ di utilizzare solamente

Tabella 4.8: Test con 280, 600 e 2500 paragrafi per il modello 28

num	***n-doc***	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
28	280	0.51	0.49	0.59	0.23	0.46
28	600	0.61	0.56	0.64	0.29	0.50
28	2500	0.51	0.49	0.59	0.23	0.46

Tabella 4.9: Test con 280, 600 e 2500 paragrafi per il modello 30

num	***n-doc***	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
30	280	0.44	0.47	0.58	0.23	0.46
30	600	0.61	0.57	0.64	0.29	0.49
30	2500	0.60	0.55	0.63	0.29	0.48

Tabella 4.10: Test con 280, 600 e 2500 paragrafi per il modello 35

num	***n-doc***	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
35	280	0.46	0.44	0.59	0.23	0.47
35	600	0.63	0.53	0.63	0.30	0.49
35	2500	0.63	0.57	0.65	0.31	0.51

SciBERT. Tutti i valori intermedi rappresentano il grado di combinazione tra i due).

Riportiamo nelle tabelle 4.11, 4.12, 4.13, 4.14 e 4.15 i risultati ottenuti per ciascuno dei 5 modelli scelti.

A questo punto abbiamo scelto, per ogni modello, l'α che ottiene i risultati migliori con il punteggio P@5. In particolare si è scelto *0.80* per il modello *4*, *0.70* per il *27*, *0.70* per il *28*, *0.75* per il *30* e *0.70* per il *35*. Successivamente, una volta scelti questi iperparametri, abbiamo effettuato il *reranking*, testando l'iperparametro β per ogni modello anche in questo caso tra 0.4 e 1, con passo 0.05. Questo iperparametro indica invece in quale misura il sistema di IR deve considerare i risultati prodotti dal *retriever* e dal *reranker*. Nello specifico, un valore di β pari a 0 indica di utilizzare solamente il *retriever*, mentre un valore pari a 1 di utilizzare solo il *reranker*. Tutti i valori intermedi indicano una combinazione del contributo delle due parti per la formulazione dell'output finale.

Come si può evincere dalle tabelle 4.16, 4.17, 4.18, 4.19 e 4.20, il modello che ha permesso di ottenere i risultati migliori è stato il numero *35*, addestrato sfruttando il *multi similarity miner* con *epsilon* di *0.0001* e la *multi similarity loss* con $\alpha = 2$, $\beta = 100$ e base=0.5. Il *learning rate* utilizzato è di *0.00001* e il *retrieval* e il *reranking* sono stati eseguiti rispettivamente con un *alpha* di *0.70* e un *beta* di *0.55*. Gli score ottenuti sono:

- $P@5 = 0.69$
- $P@10 = 0.58$
- $nDCG@10 = 0.64$

Tabella 4.11: Score ottenuti nel processo di *retrieval* con i diversi α testati sul modello *4*

num	***alpha***	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
4	0.40	0.52	0.44	0.54	0.21	0.40
4	0.45	0.55	0.47	0.55	0.22	0.40
4	0.50	0.57	0.47	0.56	0.22	0.41
4	0.55	0.59	0.48	0.56	0.23	0.41
4	0.60	0.61	0.49	0.56	0.23	0.42
4	0.65	0.59	0.51	0.56	0.23	0.42
4	0.70	0.61	0.51	0.56	0.23	0.42
4	0.75	0.60	0.51	0.56	0.22	0.42
4	0.80	0.62	0.49	0.54	0.21	0.42
4	0.85	0.57	0.47	0.53	0.20	0.42
4	0.90	0.53	0.44	0.50	0.18	0.42
4	0.95	0.47	0.40	0.48	0.16	0.40
4	1	0.41	0.32	0.45	0.13	0.39

Tabella 4.12: Score ottenuti nel processo di *retrieval* con i diversi α testati sul modello *27*

num	***alpha***	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
27	0.40	0.49	0.43	0.54	0.21	0.39
27	0.45	0.49	0.45	0.56	0.22	0.40
27	0.50	0.51	0.48	0.57	0.23	0.41
27	0.55	0.52	0.48	0.58	0.24	0.42
27	0.60	0.56	0.51	0.59	0.26	0.43
27	0.65	0.59	0.51	0.61	0.27	0.45
27	0.70	0.61	0.54	0.62	0.28	0.46
27	0.75	0.60	0.55	0.63	0.28	0.47
27	0.80	0.59	0.53	0.63	0.29	0.48
27	0.85	0.57	0.52	0.63	0.28	0.49
27	0.90	0.55	0.52	0.61	0.27	0.49
27	0.95	0.50	0.47	0.58	0.24	0.49
27	1	0.43	0.43	0.55	0.22	0.48

Tabella 4.13: Score ottenuti nel processo di *retrieval* con i diversi α testati sul modello *28*

num	***alpha***	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
28	0.40	0.48	0.40	0.54	0.20	0.39
28	0.45	0.49	0.41	0.55	0.22	0.40
28	0.50	0.49	0.43	0.56	0.23	0.41
28	0.55	0.55	0.47	0.58	0.24	0.42
28	0.60	0.56	0.48	0.59	0.25	0.43
28	0.65	0.61	0.48	0.60	0.26	0.44
28	0.70	0.63	0.51	0.61	0.27	0.46
28	0.75	0.62	0.55	0.62	0.28	0.47
28	0.80	0.62	0.55	0.63	0.28	0.48
28	0.85	0.57	0.53	0.62	0.28	0.49
28	0.90	0.53	0.52	0.61	0.26	0.49
28	0.95	0.51	0.47	0.60	0.24	0.49
28	1	0.45	0.44	0.56	0.21	0.48

Tabella 4.14: Score ottenuti nel processo di *retrieval* con i diversi α testati sul modello *30*

num	***alpha***	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
30	0.40	0.49	0.43	0.54	0.21	0.39
30	0.45	0.51	0.45	0.56	0.22	0.40
30	0.50	0.54	0.46	0.57	0.23	0.41
30	0.55	0.56	0.48	0.58	0.25	0.42
30	0.60	0.58	0.50	0.60	0.26	0.43
30	0.65	0.61	0.51	0.61	0.27	0.44
30	0.70	0.62	0.52	0.62	0.28	0.46
30	0.75	0.63	0.52	0.63	0.29	0.47
30	0.80	0.61	0.53	0.63	0.29	0.48
30	0.85	0.61	0.54	0.63	0.29	0.49
30	0.90	0.55	0.49	0.61	0.27	0.49
30	0.95	0.52	0.48	0.59	0.25	0.49
30	1	0.49	0.44	0.56	0.22	0.48

Tabella 4.15: Score ottenuti nel processo di *retrieval* con i diversi α testati sul modello *35*

num	***alpha***	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
35	0.40	0.47	0.43	0.55	0.21	0.40
35	0.45	0.48	0.46	0.56	0.22	0.41
35	0.50	0.50	0.49	0.57	0.24	0.42
35	0.55	0.53	0.50	0.59	0.25	0.43
35	0.60	0.58	0.53	0.60	0.26	0.44
35	0.65	0.62	0.56	0.62	0.28	0.46
35	0.70	0.64	0.57	0.63	0.29	0.48
35	0.75	0.63	0.57	0.64	0.30	0.49
35	0.80	0.61	0.58	0.65	0.30	0.51
35	0.85	0.61	0.56	0.65	0.30	0.52
35	0.90	0.59	0.54	0.65	0.30	0.53
35	0.95	0.52	0.50	0.63	0.28	0.53
35	1	0.51	0.45	0.61	0.25	0.52

Tabella 4.16: Score ottenuti nel processo di *reranking* con i diversi β testati sul modello *4*

num	***alpha***	***beta***	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
4	0.80	0.40	0.57	0.49	0.54	0.21	0.42
4	0.80	0.45	0.59	0.49	0.54	0.21	0.42
4	0.80	0.50	0.59	0.49	0.54	0.21	0.42
4	0.80	0.55	0.59	0.49	0.54	0.21	0.42
4	0.80	0.60	0.59	0.49	0.54	0.21	0.42
4	0.80	0.65	0.60	0.49	0.54	0.21	0.42
4	0.80	0.70	0.59	0.49	0.54	0.21	0.42
4	0.80	0.75	0.61	0.49	0.54	0.22	0.42
4	0.80	0.80	0.61	0.49	0.54	0.22	0.42
4	0.80	0.85	0.60	0.49	0.54	0.21	0.42
4	0.80	0.90	0.63	0.49	0.54	0.22	0.42
4	0.80	0.95	0.62	0.49	0.54	0.22	0.42
4	0.80	1	0.62	0.49	0.54	0.21	0.42

Tabella 4.17: Score ottenuti nel processo di *reranking* con i diversi β testati sul modello *27*

num	***alpha***	***beta***	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
27	0.70	0.40	0.67	0.57	0.63	0.28	0.46
27	0.70	0.45	0.67	0.57	0.63	0.28	0.46
27	0.70	0.50	0.67	0.57	0.63	0.28	0.46
27	0.70	0.55	0.67	0.56	0.63	0.28	0.46
27	0.70	0.60	0.65	0.57	0.63	0.28	0.46
27	0.70	0.65	0.67	0.56	0.63	0.28	0.46
27	0.70	0.70	0.66	0.56	0.62	0.28	0.46
27	0.70	0.75	0.67	0.55	0.62	0.28	0.46
27	0.70	0.80	0.66	0.55	0.62	0.28	0.46
27	0.70	0.85	0.65	0.54	0.62	0.28	0.46
27	0.70	0.90	0.64	0.55	0.62	0.28	0.46
27	0.70	0.95	0.63	0.55	0.62	0.28	0.46
27	0.70	1	0.61	0.54	0.62	0.28	0.46

Tabella 4.18: Score ottenuti nel processo di *reranking* con i diversi β testati sul modello *28*

num	***alpha***	***beta***	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
28	0.70	0.40	0.64	0.54	0.62	0.27	0.46
28	0.70	0.45	0.64	0.54	0.62	0.27	0.46
28	0.70	0.50	0.64	0.54	0.62	0.27	0.46
28	0.70	0.55	0.64	0.54	0.62	0.27	0.46
28	0.70	0.60	0.64	0.54	0.62	0.27	0.46
28	0.70	0.65	0.64	0.54	0.62	0.27	0.46
28	0.70	0.70	0.64	0.53	0.62	0.27	0.46
28	0.70	0.75	0.63	0.53	0.62	0.27	0.46
28	0.70	0.80	0.63	0.52	0.61	0.27	0.46
28	0.70	0.85	0.63	0.52	0.61	0.27	0.46
28	0.70	0.90	0.63	0.53	0.61	0.27	0.46
28	0.70	0.95	0.64	0.52	0.61	0.27	0.46
28	0.70	1	0.63	0.51	0.61	0.27	0.46

Tabella 4.19: Score ottenuti nel processo di *reranking* con i diversi β testati sul modello *30*

num	***alpha***	***beta***	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
30	0.75	0.40	0.58	0.53	0.63	0.29	0.47
30	0.75	0.45	0.59	0.54	0.63	0.29	0.47
30	0.75	0.50	0.61	0.54	0.63	0.29	0.47
30	0.75	0.55	0.61	0.54	0.63	0.29	0.47
30	0.75	0.60	0.63	0.54	0.63	0.29	0.47
30	0.75	0.65	0.63	0.54	0.63	0.29	0.47
30	0.75	0.70	0.61	0.54	0.63	0.29	0.47
30	0.75	0.75	0.62	0.54	0.63	0.29	0.47
30	0.75	0.80	0.63	0.54	0.63	0.29	0.47
30	0.75	0.85	0.63	0.54	0.63	0.29	0.47
30	0.75	0.90	0.64	0.55	0.63	0.29	0.47
30	0.75	0.95	0.63	0.54	0.63	0.29	0.47
30	0.75	1	0.63	0.52	0.63	0.29	0.47

Tabella 4.20: Score ottenuti nel processo di *reranking* con i diversi β testati sul modello *35*

num	***alpha***	***beta***	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
35	0.70	0.40	0.68	0.56	0.64	0.29	0.47
35	0.70	0.45	0.67	0.57	0.64	0.29	0.47
35	0.70	0.50	0.67	0.57	0.64	0.29	0.47
35	0.70	0.55	0.69	0.58	0.64	0.29	0.47
35	0.70	0.60	0.68	0.57	0.64	0.29	0.47
35	0.70	0.65	0.68	0.57	0.64	0.29	0.47
35	0.70	0.70	0.67	0.57	0.64	0.29	0.47
35	0.70	0.75	0.66	0.57	0.64	0.29	0.48
35	0.70	0.80	0.67	0.57	0.64	0.29	0.48
35	0.70	0.85	0.67	0.56	0.63	0.29	0.47
35	0.70	0.90	0.67	0.56	0.63	0.29	0.48
35	0.70	0.95	0.65	0.57	0.63	0.29	0.48
35	0.70	1	0.64	0.57	0.63	0.29	0.48

- $MAP = 0.29$
- $Bpref = 0.47$

I risultati ottenuti sono complessivamente soddisfacenti, ma sarebbe sicuramente interessante testare se, addestrando il modello su un numero maggiore di paragrafi e documenti, fosse possibile ottenere degli score ancora migliori. Tuttavia questo richiederebbe schede grafiche di ultima generazione, ad oggi difficilmente reperibili sul mercato internazionale a prezzi accessibili, con grandi disponibilità in termini di memoria e capacità di calcolo.

4.3 Confronto con lo stato dell'arte

Tabella 4.21: Tabella di confronto tra CO-Search (stato dell'arte) e il nostro miglior modello

modello	***P@5***	***P@10***	***nDCG@10***	***MAP***	***Bpref***
CO-Search	0.83	0.79	0.72	0.49	0.52
Sistema IR con SciBERT	0.69	0.58	0.64	0.29	0.47

Prima di osservare la tabella 4.21, che confronta i risultati raggiunti dal sistema di IR *Co-Search* (stato dell'arte) con quelli del nostro sistema, in merito al primo round della *TREC-COVID Challenge*, è necessario considerare alcune premesse. Innanzitutto il nostro modello è stato addestrato su solamente 600 documenti, per via dei limiti delle risorse che abbiamo avuto a disposizione, mentre *CO-Search* è addestrato sull'intero dataset che conta più di 40,000 paper scientifici. In secondo luogo il sistema di IR che costituisce lo stato dell'arte conta un numero molto elevato di parametri da addestrare, maggiore di 110 miliardi, mentre il nostro modello ne conta circa 100 milioni e risulta dunque molto più leggero. Inoltre, il modello alla base di CO-Search è costituito da due parti, una addestrata su task di question answering utilizzando SiameseBERT e GTP-2, menre l'altra addestrata su task di text summarization e utilizzante un modello non meglio specificato. Il modello del nostro sistema, invece, è solamente costituito da SciBERT. Infine, gli sviluppatori del sistema CO-Search hanno addestrato alcune reti neurali utilizzando dataset labellati mentre noi, al contrario, abbiamo sfruttato solamente il *self-supervised learning*. Dunque, sebbene i risultati raggiunti dal nostro sistema siano inferiori rispetto allo stato dell'arte abbiamo dimostrato che è possibile ottenere score apprezzabili anche con un sistema molto più leggero che probabilmente, con le risorse adeguate

per effettuarne l'addestramento, potrebbe avvicinarsi ulteriormente ai punteggi performati da *CO-Search*.

Capitolo 5

Conclusioni e sviluppi futuri

5.1 Discussione dei risultati

In questo progetto di tesi abbiamo proposto un approccio self-supervised per la *TREC-COVID Challenge*, una competizione in campo AI che è nata con lo scoppio della pandemia globale. Questa challenge prevedeva la creazione di sistemi di *information retrieval* efficienti che fossero in grado di individuare informazioni utili per la comunità scientifica all'interno del grande quantitativo di paper rilasciati sul *Covid-19* e sui suoi precursori, in modo da cercare di individuare possibili cure, vaccini e trattamenti. Tutti questi dati sono raccolti all'interno del dataset dinamico non labellato CORD-19, utilizzato poi per la competizione. Lo stato dell'arte su TREC-COVID ha utilizzato dataset labellati affini come MSMarco o HotpotQA, per addestrare i modelli di IR. L'utilizzo di tali dataset è un problema ben noto per l'applicazione del deep learning poiché solo pochi domini ne hanno uno e crearne di nuovi richiede molto tempo e una grande interazione umana, che si traduce in un costo economico elevato. Pertanto, approci unsupervised, o self -supervised dove queste label non sono necessarie, stanno diventando il focus di un numero crescente di ricercatori. Anche noi abbiamo deciso di adottare un approccio di tipo *self-supervised* per l'addestramento tramite deep metric learning, di un modello di rete neurale basato sul transformer *SciBERT*, da utilizzare poi all'interno di un sistema di IR.

Abbiamo quindi optato per addestrare il modello sfruttando le tecniche di deep metric learning, nate per la computer vision e con lo scopo di rappresentare nello spazio di embeddings su posizioni vicine immagini tra loro simili, mentre su posizioni lontane immagini differenti. Basandoci sul presupposto che, dato che immagini e testo vengono rappresentate entrambe attraverso vettori di numeri reali chiamati embeddings, questo approccio poteva essere adattato anche al natural language processing, in quanto i modelli addestrati

con queste tecniche sarebbero stati in grado di rappresentare su posizioni vicine elementi testuali semanticamente simili e su posizioni lontane elementi di testo semanticamente differenti. Abbiamo deciso di creare un train set composto da 5 paragrafi di ogni documento, addestrando il modello a posizionare vicini quelli provenienti dal medesimo articolo e allontanare tra loro quelli che appartenevano ad articoli diversi. Abbiamo dunque addestrato il modello di transformer SciBERT con loss tipiche del deep metric learning, in particolare *Triplet Loss*, *Angular Loss*, *Lifted Structure Loss* e *Multi Similarity Loss*, osservando che quest'ultima è quella che performa meglio. Successivamente abbiamo testato le versioni migliori del modello SciBERT su un sistema di *information retrieval*, costituito da un *indexer*, un *retriever* e un *reranker*. Una volta individuati gli iperparametri migliori abbiamo prodotto il risultato finale che, sebbene non riesca a raggiungere lo stato dell'arte sul task in oggetto, rappresentato dal sistema di IR *CO-Search*, ha permesso comunque di raggiungere dei risultati soddifacenti, dimostrando la validità dell'idea alla base del progetto.

Nel confronto con lo stato dell'arte bisogna inoltre tenere conto del fatto che: (i) il nostro modello è stato addestrato solamente su 600 documenti non labellati provenienti da CORD19, dei quali abbiamo considerato solamente i primi 5 paragrafi, per via delle limitate risorse hardware a disposizione, mentre i competitors come CO-Search hanno utilizzato l'intero corpus CORD19, più altri dataset precedentemente labellati come MSMarco, HotpotQA e SQuAD; (ii) la nostra soluzione utilizza una sola rete neurale nella versione base, mentre CO-Search è costituito da modelli molto più complessi, pesanti e con un numero di parametri da addestrare superiore di alcuni ordini di grandezza rispetto a quello da noi utilizzato.

Infine, con il nostro lavoro, abbiamo identificato che su questo dominio specifico la Multi-Similarity Loss supera nettamente le altre metric loss utilizzate per via della sua capacità di modellare su vari livelli gli elementi simili, estraendone una migliore informazione. Probabilmente la diversa struttura dei paragrafi di un documento fa si che la similarità tra essi non sia sempre così netta e chiara. Riteniamo che per questo, una loss capace di apprezzare e sfruttare queste variazioni nella similarità come la MSL, funzioni meglio di altre loss che invece trascurano maggiormente questo aspetto.

5.2 Sviluppi futuri

Questo progetto presenta sicuramente molti spunti per possibili sviluppi futuri. In primo luogo sarebbe molto interessante testare anche altre funzioni di loss e altri miner tipici del deep metric learning per osservare se è effettivamente possibile creare una configurazione che permetta di raggiungere un risultato

più performante su questo task rispetto all'utilizzo di *Multi Similarity Loss* e *Multi Similarity Miner*. In secondo luogo, sarebbe utile riuscire a riaddestrare i modelli con risorse hardware superiori e in particolare con GPU ad elevata memoria e capacità di calcolo. In questo caso si potrebbe testare se, considerando un numero di paragrafi per documento superiore al 5 (massimo che per noi è stato possibile raggiungere per via dei limiti delle GPU fornite da Google Colaboratory) o addirittura considerando tutti i paragrafi di ciascun documento, fosse possibile ottenere risultati ancora più promettenti e avvicinarsi ulteriormente allo stato dell'arte costituito da *CO-Search* sul primo round della *TREC-COVID Challenge* o addirittura superarlo. Qualora fosse possibile considerare tutti i paragrafi di ogni documento sarebbe inoltre interessante osservare i risultati raggiunti dal modello sul dataset completo (oltre 40,000 documenti), anziché su solamente 600.

Ringraziamenti

La prima persona che voglio ringraziare è il Prof. Gianluca Moro per avermi permesso di svolgere questa tesi e aver alimentato il mio interesse per questa splendida disciplina, sin da quando ho seguito le lezioni del corso di *Programmazione di Applicazioni Data Intensive*. In secondo luogo vorrei ringraziare il Dott. Lorenzo Valgimigli per i preziosi consigli e il supporto fornito durante lo sviluppo del progetto. Ci tengo poi a dedicare questo lavoro ai miei genitori, a mio fratello e ai miei nonni, che mi hanno sempre supportato in ogni mia decisione e in generale anche a tutti gli amici che mi sono stati vicino e a tutti quelli con i quali ho trascorso questo percorso della mia vita.

Bibliografia

[1] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[2] Donald Olding Hebb. *The organisation of behaviour: a neuropsychological theory*. Science Editions New York, 1949.

[3] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[4] Minsky Marvin and A Papert Seymour. Perceptrons, 1969.

[5] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[8] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676*, 2019.

[9] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[10] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[11] Stephen Robertson and Hugo Zaragoza. *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc, 2009.

[12] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. In *Linear algebra*, pages 134–151. Springer, 1971.

[13] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[14] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

[15] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, 2013.

[16] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[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. *arXiv preprint arXiv:1804.07461*, 2018.

[19] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[20] Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. Swag: A large-scale adversarial dataset for grounded commonsense inference. *arXiv preprint arXiv:1808.05326*, 2018.

[21] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[22] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[23] Feng Wang and Huaping Liu. Understanding the behaviour of contrastive loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2495–2504, 2021.

[24] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International workshop on similarity-based pattern recognition*, pages 84–92. Springer, 2015.

[25] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4004–4012, 2016.

[26] Jian Wang, Feng Zhou, Shilei Wen, Xiao Liu, and Yuanqing Lin. Deep metric learning with angular loss. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2593–2601, 2017.

[27] Xun Wang, Xintong Han, Weilin Huang, Dengke Dong, and Matthew R Scott. Multi-similarity loss with general pair weighting for deep metric learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5022–5030, 2019.

[28] Lucy Lu Wang, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Darrin Eide, Kathryn Funk, Rodney Kinney, Ziyang Liu, William Merrill, et al. Cord-19: The covid-19 open research dataset. *ArXiv*, 2020.

[29] Kirk Roberts, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, Kyle Lo, Ian Soboroff, Ellen Voorhees, Lucy Lu Wang, and William R Hersh. Searching for scientific evidence in a pandemic: An overview of trec-covid. *arXiv preprint arXiv:2104.09632*, 2021.

[30] Andre Esteva, Anuprit Kale, Romain Paulus, Kazuma Hashimoto, Wenpeng Yin, Dragomir Radev, and Richard Socher. Co-search: Covid-19 information retrieval with semantic search, question answering, and abstractive summarization. *arXiv preprint arXiv:2006.09595*, 2020.

[31] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[32] Peilin Yang, Hui Fang, and Jimmy Lin. Anserini: Enabling the use of lucene for information retrieval research. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1253–1256, 2017.

[33] Edward Loper and Steven Bird. Nltk: The natural language toolkit. *arXiv preprint cs/0205028*, 2002.

[34] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[36] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[37] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. Pytorch metric learning. *arXiv preprint arXiv:2008.09164*, 2020.

[38] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.