Alma Mater Studiorum · Università di Bologna
Campus di Cesena

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

ANALISI E SPERIMENTAZIONE DI APPROCCI DI INFORMATION RETRIEVAL NEURALE

Relatore

Prof. Gianluca Moro

Presentata da

Veronica Biancacci

Co-relatore

Dott. Ing. Lorenzo Valgimigli

Anno Accademico 2019/2020

Machines take me by surprise
with a great frequency.

- Alan Turing

Introduzione

Con l'aumentare della quantità di dati testuali digitalizzati presenti nel Web e non solo, aumenta sempre di più anche il bisogno di creare sistemi di **Information Retrieval**, capaci di rappresentare, memorizzare e soprattutto recuperare le informazioni non strutturate, sulla base di una determinata richiesta.

Dai primi sistemi più elementari, basati su indicizzazione di documenti, si sono sviluppate negli ultimi anni tecnologie considerate 'intelligenti'. Queste fanno uso di tecniche di *Machine Learning* e *Deep Learning*, le quali hanno preso piede nell'ultimi decenni grazie alla grande quantità di dati di cui possiamo disporre ora e ai nuovi sistemi di calcolo parallelo basati su GPU.

Mentre i primi modelli basano la loro ricerca sull'analisi lessicale delle query in input, gli algoritmi di Machine Learning iniziano a modellare anche la semantica delle frasi, calcolando la frequenza che le parole hanno nel testo. La vera rivoluzione arriva però con l'introduzione delle Reti Neurali, le quali riescono a modellare la conoscenza degli input e le relazioni semantiche in maniera quasi umana.

In particolar modo queste tecniche di **Natural Language Processing** basate su Intelligenza Artificiale, si sono evolute molto proprio negli ultimi anni, e continuano a farlo tutt'ora, grazie all'introduzione, prima delle *Reti Neurali Ricorrenti LSTM* e poi del *Meccanismo di Attenzione*, che hanno segnato una rivoluzione nei metodi di rappresentazione del testo.

In questa tesi si vanno ad analizzare proprio questi sistemi, a partire dai più semplici, fino ad arrivare a quelli impiegati nella creazione di modelli di

Information Retrieval Neurale, i quali utilizzano tecnologie all'avanguardia, che hanno raggiunto lo stato dell'arte in molti task di NLP.

Verrà presentato infine lo sviluppo di un sistema di Information Retrieval Neurale finalizzato all'esecuzione di task sul dataset della competizione Kaggle *"COVID-19 Open Research Dataset Challenge (CORD-19)"*.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Information Retrieval

1.1 Cos'è l'Information Retrieval?

Con **Information Retrieval** si indica l'insieme delle tecniche utilizzate per la rappresentazione, memorizzazione, organizzazione e recupero di un insieme di documenti contenenti informazioni.

È un sotto ambito del **Natural Language Processing** (NLP), un campo di ricerca dell'Intelligenza Artificiale che abbraccia diversi settori, dall'*Informatica* alla *Linguistica*. Questa disciplina ha come scopo l'analisi, la comprensione e la rappresentazione del linguaggio naturale umano, scritto o parlato.

A questo scopo utilizza tecniche di **Machine Learning** e **Deep Learning**, capaci di mappare parole, frasi o documenti in uno spazio vettoriale ad alta dimensionalità detto spazio latente dal quale emergono le similarità semantiche.

Nello specifico, l'obbiettivo dell'Information Retrieval è quello di creare sistemi capaci di soddisfare bisogni informativi a partire da una collezione di documenti digitali e un'interrogazione da parte di un utente, detta query.

Esistono diverse metodologie utilizzate per recuperare le informazioni richieste, ma indipendentemente dall'applicazione utilizzata, il funzionamento di base del sistema è quello di confrontare la query di input con i documenti di cui si dispone e creare un rank di questi, in modo da selezionare e restituire

quelli più rilevanti, i quali soddisfano l'interrogazione in questione.

Le differenze tra i diversi modelli risiedono principalmente nel metodo di rappresentazione dei dati e in quello di confronto tra di essi.

1.2 I primi sistemi

L'idea di memorizzare le informazioni in un computer e recuperarle in seguito, in modo veloce e flessibile, in base alla loro rilevanza, è stata introdotta per la prima volta nel 1945 da Vannevar Bush nell'articolo *"As We May Think"*[**1**]. L'ingegnere americano afferma infatti in questo saggio:

> "When data of any sort are placed in storage, they are filed alphabetically or numerically, and information is found (when it is) by tracing it down from subclass to subclass."

> "Quando i dati di un qualsiasi tipo vengono archiviati, questo viene fatto in ordine alfabetico o numerico e le informazioni vengono recuperate (quando sono presenti) rintracciandole da sottoclasse a sottoclasse."

Questo mostra come la sua idea di recupero di informazioni si basasse sull'indicizzazione dei dati archiviati, metodo ampiamente utilizzato fin dai primi sistemi di Information Retrieval implementati.

Queste tecniche furono studiate già dagli anni '70, ma il grande interesse per questa disciplina crebbe solo negli anni '90, grazie alla diffusione di Internet e dei motori di ricerca.

1.3 Full-Text Search

Il **Full-Text Search** (FTS) si riferisce ad un insieme di tecniche utilizzate per il *Text Retrieval*, ossia l'Information Retrieval su dati testuali.

Questi metodi richiedono l'estrazione e l'indicizzazione dei termini in ogni documento, preceduti da tecniche di NLP per il pre-processing del testo.

1.3.1 Tecniche di pre-processing

Essendo il linguaggio naturale ambiguo e destrutturato, questo ha bisogno di essere organizzato ed etichettato con informazioni utili al calcolatore per facilitare l'analisi linguistica.

Sono diverse le **tecniche di pre-processing** che possono essere utilizzate per strutturare il testo. Le più comuni e mirate ad estrarre la sintassi del testo sono:

- TOKENIZATION: processo che divide il documento preso in input in più token (frasi, parole, ecc.)
- RIMOZIONE ELEMENTI NON NECESSARI: quali punteggiatura e tag HTML, stopwords (articoli, congiunzioni e preposizioni, ecc.)
- CASEFOLDING: processo di conversione del testo in minuscolo, in modo tale da poter riconoscere le stringhe che rappresentano la stessa parola.
- LEMMATIZATION: processo di riduzione della parola alla sua forma canonica, il lemma, che è la forma in cui una parola è registrata nel dizionario.
- STEMMING: il processo di riduzione della parola alla sua radice morfologica, il tema, eliminando eventuali desinenze e prefissi.

1.3.2 Precision e Recall

La FTS è un compromesso tra esattezza e completezza.

Mentre da una parte si vogliono recuperare solo le informazioni che soddisfano il bisogno informativo dell'utente, d'altra parte si vuole fornire un nu-

mero di documenti più grande possibile, tra quelli ritenuti rilevanti rispetto la query in input.

Queste due caratteristiche di un sistema di Infromation Retrieval possono essere misurate attraverso due metriche.

La *precision* indica l'esattezza, ossia la percentuale di documenti rilevanti tra quelli restituiti ed è calcolata come segue:

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

La *recall* invece indica la completezza, ossia la percentuale di documenti rilevanti restituiti rispetto a tutti i documenti rilevanti presenti nella collezione ed è calcolata come segue:

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

1.3.3 Modello Booleano

Uno dei modelli di Full-Text Search più semplici da implementare è il **modello booleano**, basato sulle operazioni su insiemi. I documenti vengono classificati in rilevanti e non, in base ad un'interrogazione booleana.

Ad ogni termine della query di input viene associato un insieme di documenti rilevanti, quelli che contengono il termine in esame, ad esempio attraverso una struttura indicizzata chiamata *inverted index*, ovvero una matrice che indica per ogni termine i documenti in cui esso compare.

I diversi insiemi possono poi essere combinati tra di loro attraverso operatori logici, quali AND e OR.

Questa tecnica restituisce quindi, a differenza degli altri modelli che andremo ad analizzare, un insieme di documenti ritenuti rilevanti, avvolte in numero troppo alto o troppo basso, senza nessun ordine significativo.

1.3.4 Vector Space Model

Il **Vector Space Model** è un metodo algebrico di Full-Text Search, in cui il corpus viene rappresentato tramite una matrice termini-documenti DxN, con D numero di parole distinte nel corpus e N numero di documenti.

Ogni parola viene quindi mappata in un vettore N-dimensionale che riporta la rilevanza che questa ha in ogni documento, la quale può essere calcolata in diversi modi.

Bag of Words

Il modello **Bag of Word** rappresenta i termini attraverso un multiset che descrive il numero di occorrenze che questi hanno in ciascun documento.

Si otterrà quindi un vettore sparso per ogni parola, con 0 in corrispondenza delle celle che rappresentano i documenti che non contengono la determinata parola, altrimenti n, con $n > 0$, il quale indica il numero di volte che la parola è presente nel documento.

Questa rappresentazione restituisce una lista di documenti classificati dal più rilevante al meno rilevante, in base al numero di occorrenze dei termini della query all'interno dei documenti.

Questa soluzione non è sempre delle migliori. Basti pensare che se un testo è molto più lungo di un altro, ha molte più probabilità di contenere la parola target in numero maggiore e di conseguenza verrà classificato con uno score più alto dell'altro documento, anche se non necessariamente più rilevante.

Per risolvere questo problema si può introdurre una nuova metrica per il calcolo della rilevanza delle parole nei documenti: il TF-IDF.

TF-IDF

TF-IDF (*term frequency–inverse document frequency*) è una funzione di peso utilizzata per misurare l'importanza di un termine all'interno di un documento di una collezione.

Questo valore cresce in modo direttamente proporzionale alla frequenza che la parola ha rispetto al documento e inversamente proporzionale alla frequenza rispetto alla collezione e può essere espresso come segue:

$$(\text{tf-idf})_{\text{i,j}} = \text{tf}_{\text{i,j}} \times \text{idf}_{\text{i}}$$

con i il termine e j il documento.

L'idea di base è che la parola aumenta di importanza all'aumentare del numero di volte che è contenuta nel documento, ma allo stesso tempo se la sua frequenza all'interno della collezione è molto alta, il termine è molto comune e di conseguenza perde di rilevanza all'interno del singolo documento.

Il *TF* (frequenza del termine nel documento) è calcolato come:

$$\text{tf}_{\text{i,j}} = \frac{n_{i,j}}{|d_j|}$$

dove $n_{i,j}$ è il numero di occorrenze del termine nel documento e $|d_j|$ è la dimensione del documento, espressa come numero di termini al suo interno.

L'*IDF* (l'inverso della frequenza del termine nella collezione) è espresso come:

$$\text{idf}_{\text{i}} = \log \frac{|D|}{|\{d : i \in d\}|}$$

dove $|D|$ è il numero di documenti nella collezione, mentre al denominatore abbiamo il numero di documenti che contengono il termine i.

Similarità Coseno

Una volta rappresentate le parole e i documenti tramite vettori, per fare Information Retrieval e recuperare i documenti rilevanti a partire da un'interrogazione, si rappresenta la query nello stesso spazio vettoriale degli altri documenti e si calcola la distanza che il primo ha con tutti gli altri vettori ottenuti.

In questo modo si ottiene uno valore per ogni documento, che rappresenta la distanza del corrispettivo vettore con quello di input, il quale va ad indicare la sua rilevanza.

Una funzione che è possibile utilizzare per calcolare la distanza tra vettori è la **similarità coseno**, la quale calcola il coseno dell'angolo tra i due vettori, ossia il prodotto scalare tra i due, scalato per le loro norme:

$$\text{similarity(A, B)} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}.$$

con A e B i vettori corrispondenti ai due documenti presi in analisi e θ l'angolo compreso tra i due vettori.

Più i vettori saranno simili, più θ sarà piccolo ed il valore della similarità tra i due sarà alto.

1.4 Conoscenza Semantica

Fin'ora abbiamo analizzato tutti modelli che basano il loro meccanismo sulla frequenza che le parole hanno nei documenti, senza andare ad analizzare il significato dei termini o le dipendenze che hanno tra di loro.

Infatti le tecniche di pre-processing viste in precedenza mirano ad analizzare solo la sintassi e la struttura del testo e i modelli analizzati perdono un'informazione fondamentale come quella dell'ordine delle parole .

Per ottenere una comprensione più completa del significato di un testo bisogna quindi utilizzare tecniche per la creazione di *Word Embeddings* che sfruttano modelli di Deep Learning e che andremo ad analizzare nei prossimi capitoli.

Nonostante ciò si può provare ad integrare una piccola conoscenza semantica anche ai modelli appena visti, tramite diverse tecniche di pre-processamento che utilizzano conoscenze esterne per arricchire i token con informazioni sul loro significato.

Alcune di queste tecniche sono:

- POS TAGS: part of speech tagging è un processo che assegna a ogni parole il proprio valore grammaticale (articolo, nome, aggettivo, ecc.).
- NAMED ENTITY RECOGNITION: il processo che mappa ogni parola con la sua entità (persona, organizzazione, località, quantità, ecc.).

- WORD SENSE DISAMBIGUATION (WSD): è il processo che associa ad ogni parola il suo significato, che viene identificato in base al contesto in cui si trova, ossia in base alle parole vicine ad essa.

WordNet

Per recuperare le informazioni sulle entità ed i significati delle parole è possibile utilizzare dizionari esterni, come ad esempio WordNet.

WordNet è un database semantico-lessicale della lingua inglese che contiene più di 150 000 termini associati a 4 POS di base (nome, verbo, aggettivo, avverbio) e 29 tipi di relazioni semantiche tra i termini.

Le parole sono organizzate in oltre 100.000 *synset* (set di di sinonimi), insiemi di termini che hanno il medesimo significato e ogni set è associato ad una breve descrizione.

Se volessimo che il nostro modello reputasse equivalenti due termini sinonimi, poteremmo ad esempio risalire al significato di ogni parola confrontando il suo contesto con la descrizione di ogni synset (WSD), per poi andare a sostituire tutte le parole che appartengono allo sesso set di sinonimi con la stessa parola.

1.4.1 Latent Semantic Analysis

La **Latent Semantic Analisysis** (LSA), è una metodo di *Principal Component Analysis*, una tecnica il quale scopo è quello di ridurre il numero di variabili che descrivono un insieme di dati a un numero minore di variabili latenti, limitando la perdita di informazioni rilevanti, andando bensì ad rimuovere quelle che creano rumore.

La LSA applica questa tecnica alla matrice TF-IDF, riducendo la sua dimensionalità e ottenendo le relazioni semantiche latenti all'interno dei documenti, tramite l'utilizzo della *Singular Value Decomposition* (SVD), un metodo di decomposizione di matrici.

La SVD esegue la fattorizzazione di M (nxm), la matrice dei termini-documenti, per ottenere una sua approssimazione di dimensioni minori, pro-

iettando i dati in un nuovo spazio di dimensione k, il quale cattura le loro caratteristiche più rilevanti.

K, il fattore di decomposizione, è un iper-parametro e viene scelto in base ai dati di cui si dispone, andando ad ottimizzare le prestazioni.

La matrice M è fattorizzata nelle seguenti tre matrici:

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

dove U (nxk) è una matrice composta dagli autovettori destri della matrice MM^T (matrice similarità termini), Σ (kxk) è una matrice diagonale composta dagli autovalori in ordine decrescente e V^T (kxm) è la trasposta della matrice V composta dagli autovettori sinistri della matrice M^TM (matrice similarità documenti).

Tramite il nuovo spazio vettoriale creato, questo metodo è efficace per analizzare la semantica del testo e creare dei vettori di *embedding* senza l'utilizzo di tecniche di Deep Learning.

Capitolo 2

Machine Learning

In questo capitolo verranno introdotti i concetti base del **Machine Learning**, in quanto saranno necessari per una comprensione più completa del capitolo successivo sui *Word Embeddings*.

Il Machine Learning è un campo fondamentale dell'*Intelligenza Artificiale* che ha come scopo la creazione di algoritmi in grado di svolgere apprendimento automatico, ossia capaci di imparare autonomamente ad eseguire un determinato task, senza essere esplicitamente programmati per svolgerlo.

2.1 Tipologie di Apprendimento

Esistono diverse tipologie di Apprendimento che possono essere utilizzate per l'addestramento di un modello. Le principali sono due: l'*Apprendimento Supervisionato* e quello *Non Supervisionato*.

2.1.1 Apprendimento Supervisionato

Nell'**Apprendimento Supervisionato**, il training del modello viene effettuato su un set di dati classificati, ossia che forniscono anche i valori di output.

I due problemi più comuni posso essere suddivisi in: *Regressione*, dove i dati di input vengono mappati in variabili continue (valori numerici) e *Classificazione*, dove vengono mappati in variabili discrete (categorie).

Fase di Training

È proprio grazie alla conoscenza degli output che il modello riesce ad imparare il pattern da utilizzare per predire futuri output non conosciuti, scegliendo opportunamente dei pesi da associare ai dati di input per trovare la funzione che approssimi, nel caso della regressione, o separi, nel caso della classificazione, in modo migliore questi dati.

La **fase di addestramento** del modello consiste proprio nella ricerca di questi pesi, attraverso la minimizzazione della funzione di errore, trovata confrontando i valori di output reali con quelli predetti dal modello, attraverso la *discesa del gradiente*.

Preso un punto iniziale in modo randomico nella funzione di errore, l'algoritmo di discesa del gradiente prevede di aggiornare questo punto ad ogni step, fino a convergere nel minimo della funzione, secondo la seguente formula:

$$X_{k+1} = X_k - \eta \Delta f(X_k)$$

dove X_{k+1} è il nuovo punto trovato allo step $k+1$, X_k è il punto allo step precedente, $\Delta f(X_k)$ è il gradiente calcolato in quel punto e η, chiamato anche learning rate, è un iperparamentro che indica la lunghezza del passo di discesa nella funzione.

La funzione di errore più comunemente utilizzata in un problema di regressione è l'errore quadratico medio:

$$MSE = \frac{\sum_{i=1}^{m} (\widehat{y}_i - y_i)^2}{m}$$

dove m è il numero di istanze di training, y_i è la l'output reale dell'istanza i e $\widehat{y}_i$ è l'output predetto.

Nel caso della classificazione invece, come funzione di errore, viene utilizzata generalmente la logistic loss:

$$LogisticLoss = \sum_{i=1}^{m} \log(1 + e^{-y_i h_w(x_i)})$$

dove $h_w(x_i)$ è l'iperpiano di separazione predetto per l'istanza i.

La classificazione esegue una *Regressione Logistica*, la quale utilizza la funzione sigmoide per generare l'output, e come vedremo nella successiva sezione sul Deep Learning, è alla base delle unità neuronali.

2.1.2 Apprendimento Non Supervisionato

L'**Apprendimento Non Supervisionato** d'altra parte addestra il modello attraverso input non annotato, quindi non conoscendo l'output desiderato a priori. Grazie a queste tecniche il modello potrà essere addestrato su dataset molto più ampi, come nel caso dei dati testuali non etichettati, presenti in gran quantità nel web.

Come andremo infatti ad analizzare nel prossimo capitolo, molti modelli utilizzati per problemi di NLP, utilizzano l'apprendimento non supervisionato.

Un esempio di algoritmo che impiega questo apprendimento è il *Clustering*, il quale suddivide i dati di input in più gruppi, andando a massimizzare la similarità tra i dati di uno stesso gruppo e minimizzando quella tra i dati appartenenti a gruppi differenti.

2.2 Deep Learning

Nell'ultimo decennio si sono ottenuti molti miglioramenti nel campo del NLP grazie ad algoritmi di **Deep Learning** [**2**], un campo del Machine Learning specializzato nello studio delle **Reti Neurali Artificiali**.

Le prime teorie sui Neural Networks furono sviluppate già negli anni '80, ma solo negli ultimi anni sono state ampiamente utilizzate in vari campi, grazie alla grande quantità di dati di cui possiamo disporre ora e ai nuovi sistemi di calcolo parallelo basati su GPU.

2.2.1 Feed Forward Network

Vengono chiamate **Feed Forward Network** le Reti Neurali Artificiali classiche, le quali, come dice il nome stesso, non formano cicli tra le loro connessioni, bensì trasportano l'informazione solo in avanti.

La loro struttura si basa su una semplificazione delle *Reti Neurali Biologiche* e presenta un insieme di unità di calcolo interconnesse tra loro, chiamate *Neuroni* (o nodi).

Ogni singolo Neurone moltiplica gli input con i rispettivi pesi trovati in fase di training, calcola la somma pesata tra questi, aggiunge un bias e infine passa il risultato ad una funzione di attivazione non lineare che restituisce l'output finale.

Nel caso particolare in cui la funzione non lineare sia una *sigmoide*, ogni nodo esegue esattamente una *Regressione Logistica*.

Possiamo osservare la struttura di un'unità neurale in Figura 2.1.

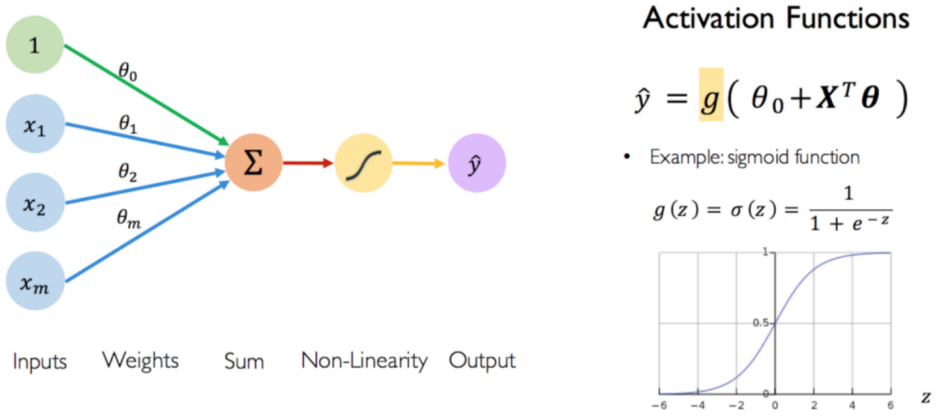

Figura 2.1: Struttura di un'unità neuronale di una Rete Neurale

I pesi da associare ad ogni elemento di input vengono opportunamente decisi durante la fase di training del modello, attraverso l'algoritmo di *Back-propagation*, il quale sfrutta la discesa del gradiente, già vista nei problemi di Machine Learning, per aggiornare e migliorare i pesi ad ogni step, in modo tale da ridurre l'errore, finché la rete non raggiunge l'accuratezza ottimale.

Come possiamo osservare in Figura 2.2 una rete neurale è composta da diversi Neuroni connessi tra loro e disposti su più layer: l'*input layer* che prende i dati in ingresso, uno o più *hidden layer* che li elaborano e l'*output layer*.

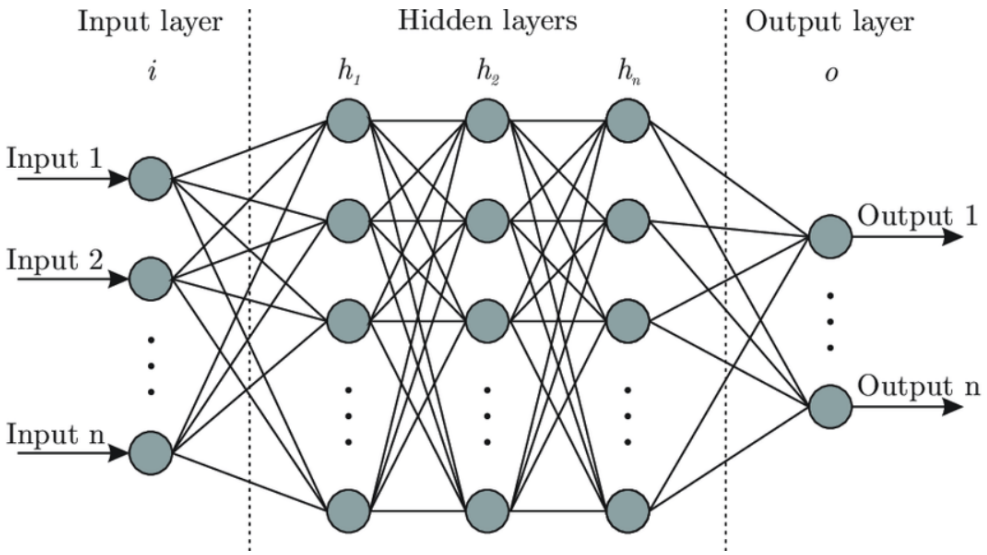

Figura 2.2: Struttura di una Rete Neurale

Grazie alla funzione di attivazione non lineare presente in ogni neurone, la rete riesce a trovare la miglior trasformazione da applicare ai dati, creando uno spazio vettoriale non lineare in cui individuare l'iperpiano di separazione, il quale riesce a classificare meglio quegli insiemi di dati che un iperpiano lineare nello spazio vettoriale originale non riuscirebbe a dividere correttamente.

Uno dei grandi limiti delle Reti Neurali Feed Forward è che il loro output dipende totalmente dagli input correnti, poiché non mantengono nessuna memoria dei precedenti input.

Questi modelli non sono quindi in grado di gestire sequenze di input, cosa di fondamentale importanza nel caso ci fosse bisogno di elaborare dipendenze a lungo termine (*long-term dependencies*) come succede con problemi di NLP, i quali prendono in input sequenze testuali.

2.2.2 Reccurrent Neural Network

Per risolvere questo problema sono state introdotte le **Reti Neurali Ricorrenti** [3] (*Reccurrent Neural Network*, RNN), strutture che a differenza delle precedenti presentano un ciclo in ogni Neurone, il quale permette di mantenere in memoria informazioni processate precedentemente, anche nelle elaborazioni successive.

Possiamo osservare una rappresentazione di una Rete Neurale Ricorrente in Figura 2.3.

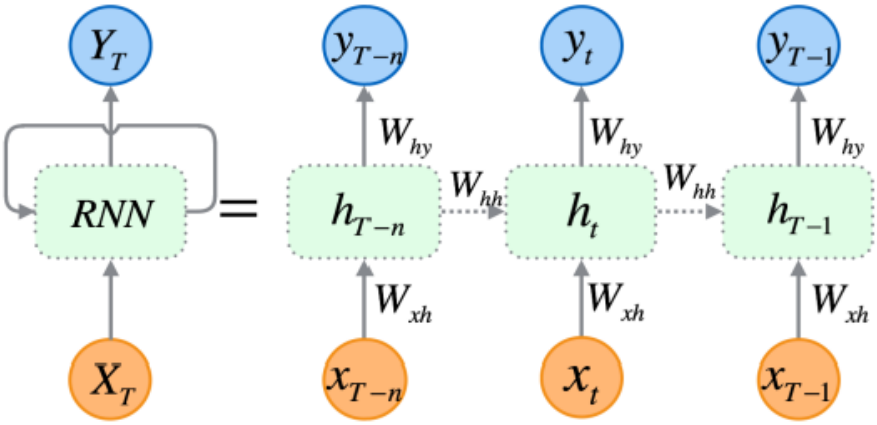

Figura 2.3: Struttura di una Rete Neurale Ricorrente

Ad ogni step il modello aggiorna il suo stato interno h_t sulla base del corrente input x_t ed il precedente stato interno h_{t-1}, secondo la seguente formula semplificata:

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

dove f_W è la funzione parametrizzata dalla matrice di pesi W, rispettivamente W_{hh} per h_{t-1} e W_{xh} per x_t.

Durante il calcolo del gradiente, a causa delle continue moltiplicazioni con le matrici W_{hh} e W_{xh}, se la sequenza in input è molto lunga ed i valori delle matrici da moltiplicare, o il valore del gradiente stesso, sono minori di 1, si incorre nel problema della *scomparsa del gradiente*. Il gradiente diventa

troppo piccolo, rendendo impossibile l'addestramento della rete, che tende a dimenticare le informazioni più lontane nella sequenza, processate nei primi step.

LSTM

Per risolvere il problema appena illustrato sono state introdotte le **LSTM** (*Long Short Term Memory*), un tipo particolare di RNN capace ad ogni step di controllare selettivamente il flusso di informazioni ricevute, mantenendole o scartandole, grazie a strutture interne chiamate *gates*, che possiamo osservare in Figura 2.4.

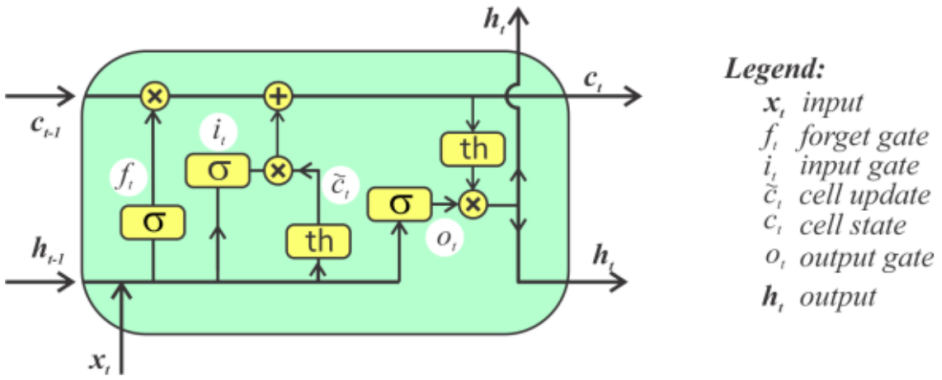

Figura 2.4: Struttura di un'unità neuronale di una Rete LSTM

Ad ogni step queste Reti:

- dimenticano le informazioni irrilevanti degli step precedenti (*forget gate*)
- memorizzano le informazioni nuove e rilevanti (*input gate*)
- aggiornano il loro stato interno sulla base delle scelte appena fatte e generano l'output (*output gate*)

Grazie alla loro struttura le LSTM riescono a gestire sequenze molto lunghe, portando avanti le informazioni rilevanti processate nei primi step,

fino agli step fiali, risolvendo il problema della memoria a breve termine delle classiche RNN.

2.3 Transfer Learning

Il **Transfer Learning** è un insieme di tecniche sviluppate recentemente e utilizzate per spostare la conoscenza appresa su un determinato dominio ad un altro dominio.

Una di queste tecniche consiste nel **Fine-Tuning**, il quale permette di prendere un modello pre-addestrato su una grande quantità di dati riguardanti un dominio più ampio e riaddestrarlo su dei dati in quantità minore, riguardanti un sotto dominio di quello originale.

Questo metodo viene ormai utilizzata in vari modelli di NLP, in quanto presenta molti vantaggi.

In primo luogo il Fine-Tuning richiede un tempo molto ridotto rispetto al Training iniziale, permettendo quindi di eseguire la parte dispendiosa una sola volta e riaddestrare il modello in modo veloce per i task specifici desiderati, come vedremo ad esempio nel caso di BERT nel prossimo capitolo.

Inoltre questo approccio risolve il problema di dataset di misure troppo ridotte. Infatti per ottenere un modello di Deep Learning con alte prestazioni bisogna allenarlo su una gran quantità di dati.

Quindi nel caso in cui non si disponga di abbastanza dati per un problema specifico, il modello può essere addestrato inizialmente su dati più generali, per poi eseguire il Fine Tuning sul task richiesto.

Capitolo 3

Word Embeddings

Attraverso la rappresentazione Vector Space Model precedentemente illustrata si ottengono vettori sparsi, chiamati anche *explicit vector*, spesso di dimensioni molto elevate, pari al numero di documenti o alla dimensione del vocabolario utilizzato.

Al contrario, i **Word Embedding** [4] sono una rappresentazione basata sul significato delle parole e generata solitamente in modo semi supervisionato, senza quindi il bisogno di avere un testo annotato in input.

Ogni parola viene mappata in un vettore, in uno spazio multi-dimensionale, generalmente di dimensione minore di una rappresentazione attraverso explicit vector, poiché indipendente dal numero di termini e documenti presenti nella collezione.

Più due parole, frasi o documenti hanno significato simile, più i vettori che li rappresentano avranno una distanza coseno minore. Due parole sono considerate semanticamente simili, se hanno un contesto simile, ossia se sono accompagnate nella frase dalle stesse parole o a sua volta da parole simili.

Anche se gli explicit vector, come dice la parola stessa, sono vettori più intuitivi e facili da visualizzare, gli *embeddings* si sono rilevati essere una rappresentazione più precisa e utile in problemi di NLP, capace di racchiudere all'interno la semantica delle parole e le relazioni tra i termini stessi.

Grazie al continuo progresso delle tecniche di *Neural Network*, anche que-

sti modelli migliorano di anno in anno, portando alla creazione di tecnologie capaci di superare le performance delle precedenti, raggiungendo lo stato dell'arte in diversi NLP task.

Abbiamo già analizzato una tecnica di Word Embeddings tramite LSA, la quale va a ridurre la dimensionalità dei vettori che rappresentano il testo, partendo dagli explicit vector calcolati tramite TF-IDF e restituendo dei vettori in un nuovo spazio vettoriale, capici di racchiudere anche la semantica del testo.

I successivi metodi che andremo ad analizzare sono tecniche Neurali, ossia che utilizzano il Deep Learning per mappare il testo di input in embeddings.

3.1 Word2Vec

Uno dei metodi più semplici e popolari di Word Embeddings Neurale è lo *Skip-gram* di **Word2Vec**. Questo modello consiste in una rete neurale costituita da un unico hidden layer di N neuroni ed addestrata, su grandi corpus di testi, a prevedere il contesto delle parole.

Riceve in input le parole rappresentate come vettori one-hot e restituisce in output vettori della stessa dimensione con i valori ad 1 in corrispondenza delle parole vicine a quella data in input.

Una volta addestrata la rete, i pesi trovati vengono utilizzati per l'embedding delle parole, che saranno rappresentate in uno spazio di dimensione N.

Possiamo osservare la rappresentazione di questa rete in Figura 3.1.

Un altro vantaggio di questo modello è la sua capacità di incorporare nel vettore le relazioni tra le parole. Infatti, coppie di parole con la stessa relazione sono rappresentate attraverso vettori con differenze simili.

È quindi capace di riconosce relazioni del tipo *"uomo sta a donna come re sta a regina"*, come si può osservare in Figura 3.2.

In seguito alla popolarità di Word2Vec, nel corso degli ultimi anni, molte sono state le architetture sviluppate con tecniche di Deep Learning per

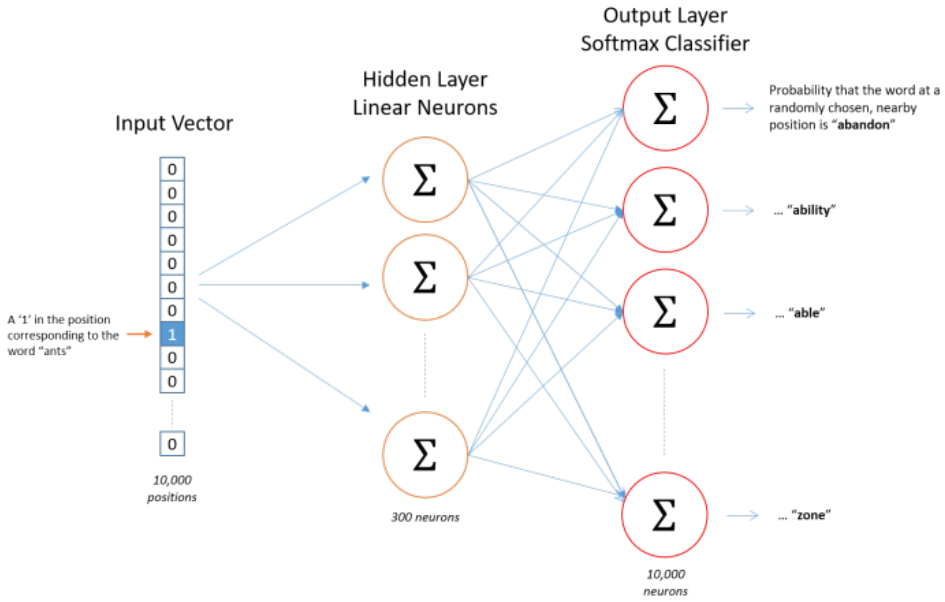

Figura 3.1: Rappresentazione modello Skip-gram di Word2Vec

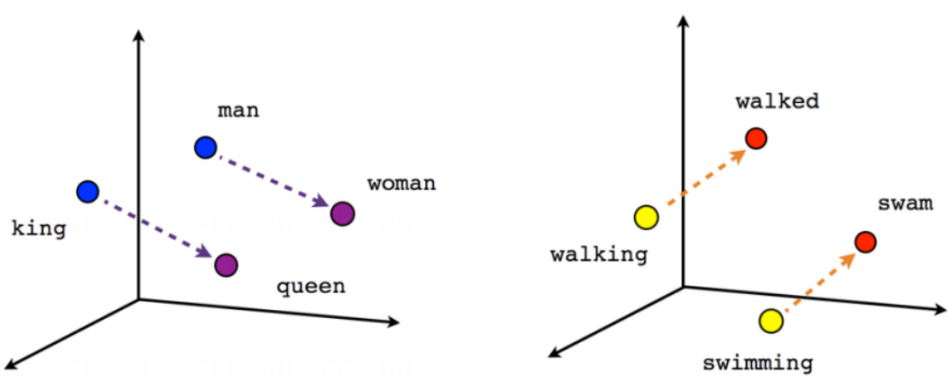

Figura 3.2: Rappresentazione delle relazioni tra i vettori in Word2Vec

generare Word Embeddings.

3.2 BERT

Uno dei modelli che ha riscontrato più successo nell'ambito del NLP negli ultimi anni e che è capace di incorporare il significato e le relazioni tra le

parole stesse tramite una rappresentazione vettoriale, è **BERT [5]** (*Bidirectional Encoder Representations from Transformers*) rilasciato da Google nel 2018.

Le novità introdotte da BERT sono varie.

Innanzi tutto BERT utilizza un modello bidirezionale. È quindi in grado di rappresentare una parola in base al contesto della frase, sia che le informazioni rilevanti si trovino a destra che a sinistra, a differenza delle architetture *left-to-right*, le quali guardano solo alle parole precedenti a quella che si sta processando.

Una delle novità principali sta nel fatto che riesce a fare ciò senza l'utilizzo di layer ricorrenti, come nel caso delle tecnologie basate su LSTM. Bensì utilizza un meccanismo chiamato Self-Attention, che verrà analizzato nei prossimi paragrafi.

3.2.1 Architettura

L'architettura di BERT, così come quella di tutti i modelli che utilizzano Self-Attention, è basata su quella del **Transformer [6, 7]**, un modello innovativao che va per la prima volta ad eliminare le Reti Neurali Ricorrenti, come le LSTM, utilizzate fino a quel momento in molti ambiti di NLP, per la loro caratteristica di mantenere le informazioni rilevanti durante il processamento di tutta la sequenza di input.

Questo caratteristica, fino a quel momento ritenuta indispensabile per una rappresentazione accurata del testo in linguaggio naturale, viene sostituito nei Transformer proprio dal meccanismo di Self-Attention, introdotto per la prima volta nel 2017 da Google nel paper *"Attention is all you need"* **[8]**.

L'architettura di BERT consiste infatti in un multilayer bidirectional encoder, il quale corrisponde all'encoder del Transformer.

L'architettura originale del Transformer è costituita da un modello *Encoder-Decoder*. Si può osservare la struttura del suo encoder, utilizzato da BERT, in Figura 3.3.

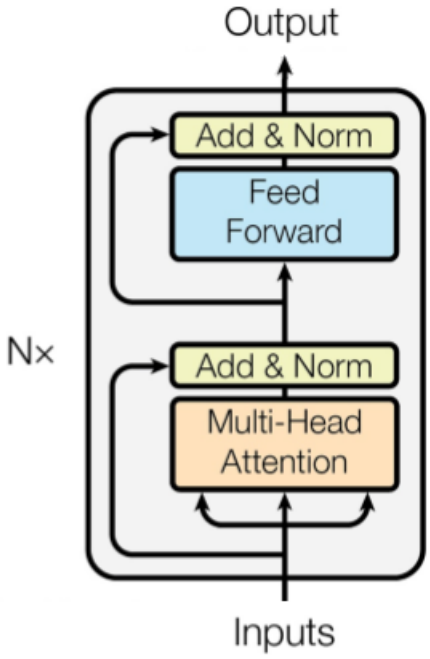

Figura 3.3: Rappresentazione architettura dell'encoder di un Transformer

Questo è formato da N layer posti in sequenza, tutti composti da un sistema di *Multi-Head Attention* e da una rete Feed Forward, entrambi seguiti da un layer di normalizzazione.

Ogni layer dell'encoder contenuto in BERT, elabora l'input e restituisce al layer successivo un vettore contenente l'ultimo stato interno calcolato, che rappresenta le informazioni rilevanti dell'input.

3.2.2 Self-Attention Mechanism

Ma a cosa serve esattamente il **meccanismo di self-attention**?

Questo permette al modello, nel momento in cui sta processando un determinato token, di andare a guardare agli altri elementi della frase di input, alla ricerca di quelli più rilevanti, i quali possono aiutare a creare un encoding migliore del token in questione.

Questo meccanismo è un metodo alternativo alle RNN per ottenere le informazioni dell'input utili per la generazione dell'output di un determinato token. Andiamo ora a vedere questo meccanismo più nel dettaglio.

La soluzione prende in input tre matrici, V (*value*), K (*key*) e Q (*query*), rispettivamente i valori dei token, che esprimono la loro rilevanza rispetto al target, le chiavi ad essi associati, che esprimono la loro direzione e la matrice dei token target. Viene calcolata la similarità tra K e Q, attraverso un prodotto scalare tra i due vettori, scalato con la radice del loro modulo, e viene passato il risultato ad una softmax e moltiplicato infine per V come mostrato nella formula seguente:

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

Attraverso il valore finale riusciamo quindi a selezionare quale sono i token a cui bisogna porre più o meno attenzione, durante il proprocessing di un determinato token, per poterlo comprende a pieno.

Questo appena esposto descrive un singolo meccanismo di self-attention. BERT in realtà impara più meccanismi, chiamati head, 12 per ogni encoder, i quali operano in parallelo. Da questi deriva il nome *Multi-Head Attention*, il quale indica il primo layer dell'encoder del Transformer, come abbiamo già visto in Figura 3.3.

Questo meccanismo multiplo permette al modello di catturare un range molto più ampio di relazioni tra le parole in input, migliorando quindi la rappresentazione finale.

3.2.3 Pre-Traning

L'ultima novità introdotta da BERT sta nelle nuove tecniche non supervisionate di pre-training utilizzate.

La prima è il *Mask Language Model* (LML), dove, al fine di addestrare il modello, vengono mascherate il 15% delle parole date in input, in modo randomico e viene addestrato il modello a prevedere il loro valore iniziale sulla base di tutta la sequenza.

La seconda invece è la *Next Sentence Prediction* (NSP), utilizzata per relazionare le frasi tra di loro. Viene fornita in input la coppia di frasi A e B, con il 50% di possibilità che B sia la frase che segue realmente A. Il modello deve prevedere se B è davvero la frase successiva o meno.

A partire dal modello di Bert pre-addestrato risultante, si possono raggiungere grandi risultati in moltissimi task NLP, procedendo con il Fine Tuning specifico per il determinato compito.

3.2.4 Rappresentazione dell'input

Andiamo ora ad analizzare come viene rappresentato l'input passato al modello. Questo consiste in una coppia di frasi, A e B, precedute da un token [CLS] e separate dal token [SEP] che viene posizionato anche come token finale.

L'input viene codificato attraverso un **Token Embedding** e viene sommato al **Segment Embedding**, che codifica l'appartenenza della parola alla frase A o B, e al **Positional Encoding**, che codificale la posizione, in quanto BERT non utilizzando le RRN non riconosce da sè la posizione delle parole nella frase.

Possiamo osservare la rappresentazione dell'input in Figura 3.4.

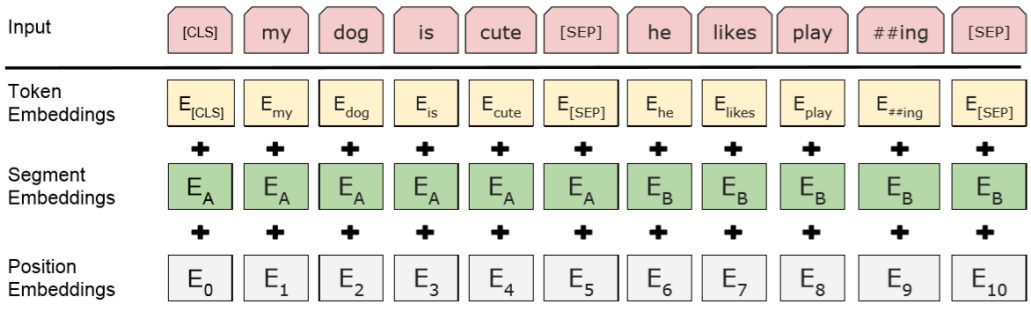

Figura 3.4: Rappresentazione dell'input passato al modello BERT

WordPiece

Come appena mostrato, BERT codifica l'input utilizzando il *Token Embedding*, il quale segmenta la frase originale in più token, secondo un vocabolario di 30.000 termini, creato attraverso l'algoritmo **WordPiece**.

Questa tecnica permette di creare un dizionario delle parole e sotto-parole più comuni del corpus di training, andando a registrare in esso le combinazioni dei caratteri che aumentano la verosimiglianza (*likelihood*) nei dati utilizzati.

Nello specifico va a creare ad ogni step una nuova combinazione di caratteri andando ad aggiungere un carattere A ad un gruppo di caratteri B già scelti in precedenza, in modo tale da aumentare la *funzione di likelihood*, una funzione di probabilità condizionata espressa come segue:

$$\mathcal{L} : b \mapsto P(A|B = b)$$

Una volta creato il dizionario, nel caso in cui una determinata parola di input non fosse contenuta in esso, questa verrà rappresentata attraverso più sotto-parole presenti nel vocabolario.

Ad esempio la parola playing in Fig. 2.3 viene segmentata nelle due sotto-parole *play* e *##ing*, dove *##* va ad indicare tutte le sotto-parole successive ad un token iniziale (in questo caso play).

È proprio durante la fase di training che BERT riesce a imparare la giusta codifica di ogni token nel corrispettivo vettore d-dimenzionale, il quale sommato al Segment Embeddings e al Positional Encoding, rappresenta l'effettiva rappresentazione presa in input dal modello.

Positional Encoding

La posizione delle parole nella sequenza di input è fondamentale per comprendere la semantica di una frase. Poichè BERT non possiede nessuna nozione innata sulla posizione e l'ordine delle parole, viene utilizzato il **Positional Encoding[9]** per integrare questa informazione.

Questo metodo basa il suo meccanismo sulle funzioni di seno e coseno a frequenze differenti, riuscendo ad incorporare nel risultato finale sia la posizione assoluta che relativa della parola. Le due funzioni utilizzate sono:

$$PE(pos, 2i) = sin(pos/10000^{2i/d_{model}})$$

$$PE(pos, 2i+1) = cos(pos/10000^{2i/d_{model}})$$

dove *pos* è la posizione assoluta della parola e d è la dimensione del vettore finale, utilizzata anche per il Token e Segment Embeddings, così da poter sommare i tre vettori tra di loro.

Il risultato ottenuto è un vettore d-dimensionale con gli elementi in posizione pari definiti dalla prima funzione, quelli in posizione dispari dalla seconda.

In questo modo otteniamo un vettore di coppie seno/coseno per ogni frequenza, la quale diminuisce all'aumentare della posizione dell'elemento nel vettore stesso.

3.3 SPECTER

SPECTER (*Scientific Paper Embeddings using Citation-informed TransformER*) è un nuovo modello di embedding di documenti scientifici, introdotto quest'anno dall'istituto di intelligenza artificiale Allen e al. attraverso il paper *"SPECTER: Document-level Representation Learning using Citation-informed Transformers"* [**10**].

La novità introdotta da questo modello è la rappresentazione vettoriale *document-level*. A differenza dei modelli precedentemente esposti, compreso BERT, che rappresentano le informazioni in base al contesto intra-document, SPECTER è capace di incorporare quello inter-document, ossia le relazioni esistenti tra i documenti stessi, sfruttando, durante il training del modello, le citazioni ad altri paper.

L'**architettura** del modello si basa su quella di BERT. Nello specifico è stato utilizzato *SciBERT* [**11**], un adattamento del modello originale, ma

addestrato su un corpus di testi scientifici. La maggior differenza dal classico BERT sta appunto nell'utilizzo di un vocabolario, che si differenzia da quello originale del 58%, creato sempre attraverso l'algoritmo *WordPiece* ma su un training set differente, contenente solo testi di ambito scientifico.

Ma come vengono utilizzate le citazioni per incorporare le relazioni tra i documenti nella loro rappresentazione vettoriale?

Il modello viene inizialmente inizializzato con i pesi trovati da SciBERT, per poi essere addestrato a rappresentare i paper scientifici in un nuovo spazio vettoriale, attraverso vettori più vicini, tanto più questi documenti sono considerati simili tra di loro. Questa similarità viene appunto calcolata utilizzando le citazioni.

Attraverso la minimizzazione della loss function il modello imparerà a rappresentare più vicini due testi, quando uno dei due contiene una citazione all'altro.

Vediamo di seguito la funzione di Loss:

$$\mathcal{L} = max\{(d(\mathcal{P}^Q, \mathcal{P}^+) - d(\mathcal{P}^Q, \mathcal{P}^-) + m), 0\}$$

dove d è la funzione che calcola la distanza tra i vettori corrispondenti ai due paper ed m è un iperparametro *loss margin*, settato empiricamente a 1.

La distanza tra i vettori è calcolata attraverso la *norma L2*:

$$d(\mathcal{P}^A, \mathcal{P}^B) = \|\mathrm{v}_A, \mathrm{v}_B\|_2$$

Questa funzione di errore è anche chiamata **triplet margin loss function**, in quanto prende in input tre paper $\mathcal{P}^Q, \mathcal{P}^+ e \mathcal{P}^-$.

$\mathcal{P}^Q$ (query paper) è il paper di cui vogliamo calcolare la rappresentazione, $\mathcal{P}^+$ (*positive paper*) è un paper citato da $\mathcal{P}^Q$, mentre $\mathcal{P}^-$ (*negative paper*) è un paper non citato da esso.

Molto importante è inoltre la scelta dei paper negativi durante la fase di training. Vengono definiti due set:

- il primo consiste semplicemente in paper non citati da $\mathcal{P}^Q$, selezionati in modo randomico

- il secondo invece consiste in paper non citati da $\mathcal{P}^Q$, ma citati da un paper che è citato a sua volta dal query paper.

I paper del secondo set, chiamati *hard negatives*, dovranno esse rappresentati come vettori più vicini a $\mathcal{P}^Q$, rispetto a quelli presenti nel primo set, in quanto hanno comunque una correlazione con il paper in questione, ma a loro volta più lontani rispetto a quelli presenti in $\mathcal{P}^+$, poichè questa correlazione è minore di quella presente in un paper citato direttamente dal query paper.

Durante la fase di **inference**, viene passato al modello un singolo paper e viene restituito in output la sua rappresentazione vettoriale.

$$\text{v} = Modello(input)$$

Per ogni paper, anche durante la fase di training, è passato in input solo il titolo e l'abstact, secondo lo stesso schema visto per BERT:

[CLS] titolo [SEP] abstract [SEP]

dove *titolo* e *abstract* sono rappresentati attraverso la segmentazione WordPiece. L'output v, ossia la rappresentazione finale del paper in input, corrisponde all'ultimo hidden state associato al token [CLS], chiamato anche *pooled output*.

Capitolo 4

Progetto

In quest'ultimo capitolo viene presentato il progetto a cui ho lavorato, il quale consiste nello sviluppo di un sistema di Information Retrieval Neurale finalizzato all'esecuzione di task sul dataset della competizione Kaggle "*COVID-19 Open Research Dataset Challenge (CORD-19)*" [**12**].

Questo dataset consiste in una raccolta di oltre 200000 documenti scientifici in lingua inglese riguardante il COVID-19. La competizione chiedeva di restituire, attraverso un modello, i documenti più rilevanti tra quelli a disposizione, in base alle diverse domande date in input, riguardanti il virus.

Per la parte iniziale di questo progetto ho lavorato sul codice dell'ingegnere Lorenzo Valgimigli, il quale mi ha fornito un modello di embedding simile a SPECTER, su cui sperimentare il dataset.

Il mio è stato un lavoro di modifica e pulizia del dataset, seguito da una fase di training e confronto tra i diversi modelli ottenuti. Ho infine implementato un test per uno specifico compito ed eseguito il relativo confronto tra i risultati ottenuti e quelli restituiti dal modello migliore in quel determinato task.

4.1 Modello utilizzato

Il modello implementato dall'ingegnere Lorenzo Valgimigli è basato sul modello SPECTER, a cui sono state apportate alcune modifiche.

Anch'esso utilizza SciBERT come modello base e lo addestra utilizzando la **triplet margin loss function**, allo stasso modo di SPECTER.

La differenza maggiore sta nel fatto che l'embedding dei documenti non viene fatto sulla base delle citazioni. Infatti $\mathcal{P}^Q$, $\mathcal{P}^+$ e $\mathcal{P}^-$ passati alla loss function sono definiti in modo differente.

Innanzi tutto $\mathcal{P}^Q$ non è un paper completo come in SPECTER (titolo + abstract), bensì corrisponde solo ad un titolo. Questo permette di risolvere un problema che si riscontra con il modello originale in fase di inference.

Infatti SPECTER è capace di restituire documenti rilevanti sulla base di un altro documento, in quanto anche in fase di inference prende in in input sia il titolo che l'abstract del paper. Questo rende impossibile risolvere task che richiedono Information Retrieval sulla base di una query composta da una singola frase, anzichè un intero documento.

Per quanto riguarda il paper positivo e quello negativo, questi sono composti da soli abstract e corrispondono rispettivamente al paper che contiene il titolo definito come $\mathcal{P}^Q$ e ad un paper scelto in modo randomico.

In questo modo il modello imparerà a rappresentare con un vettori più vicini ai titoli, gli abstract corrispondenti, mentre con vettori più lontani tutti gli altri.

In fase di inference ci aspettiamo quindi di ottenere i documenti più rilevanti rispetto una determinata query di input, nelle posizioni più alte della classifica di paper restituita.

La fase di **test** è valutata su un numero ridotto di istanze scelte in modo randomico tra quelle appartenenti al test_set.

Per ogni istanza i viene creato l'embedding del corrispettivo titolo e viene confrontato attraverso la similarità coseno con l'embedding di tutti gli abtract contenuti nel test_set. Una vola ottenuto lo score per ogni abstarct, vengono

riordinati i vettori in base al punteggio di similarità e si va a verificare in che posizione si trova l'abstarct corrispondente all'istanza i. Più questo si trova tra le prime posizioni, più la performance del modello è superiore.

4.2 Pulizia del dataset

In questa fase mi sono occupata di creare i differenti dataset su cui addestrare il modello.

Scaricato il dataset originale da Kaggle, sono stati eliminati tutti i paper duplicati e quelli contenenti abstract con numero di caratteri minori di 50, così come già implementato nel codice utilizzato.

Sono poi stati filtrati nuovamente i dati, in base al numero di caratteri nel titolo di ogni istanza.

Sono stati definiti sei dataset, contenenti titoli di lunghezza superiore a 20, 30, 40, 60, 70 e 80 caratteri, rispettivamente e a partire da essi sono stati creati i diversi set per la fase di training (60 mila istanze) e quella di test (30 mila istanze).

4.3 Training, Test e confronto

È stato addestrato il modello sui diversi train_set creati, attraverso 4 epoche ciascuno e sono poi stati confrontati tra di loro i modelli ottenuti, calcolando la loss per ognuno di essi.

Nella fase di test sono state selezionate dal test_set 100 istanze in modo randomico e passate in input al modello i titoli corrispondenti a queste. Per ognuna, è poi stata calcolata la distanza coseno con i 30000 abstract delle istanze presenti nel test_set ed è stata infine restituita in output la lista dei paper, ordinata secondo lo score della distanza calcolata.

Come già anticipato, la performance del modello è migliore, quando l'abstract corrispondente al titolo in input viene classificato nelle prime posizioni della lista.

Per confrontare i diversi modelli a cui ho lavorato, ho quindi usato una funzione di test, definita come la media delle posizioni in lista dei paper target, scalata tra 0 e 1 attraverso il numero di istanze nel test_set, il quale corrisponde alla posizione più alta in lista che il paper può ottenere.

$$loss = \frac{mean(pos)}{|test\_set|}$$

La fase di test è stata ripetuta quattro volte per ogni modello addestrato, per un totale di 400 istanze testate per ognuno di essi, scelte in modo randomico, da cui è stata calcolata la loss finale e la deviazione standard dei sei modelli.

La deviazione standard è definita come:

$$\sigma_X = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N}},$$

dove N è il numero totale di istanze testale, 400 in questo caso, x_i corrisponde alla posizione ottenuta dal modello per ogni istanza i e $\bar{x}$ è la posizione ottimale, ossia 0.

4.3.1 Risultati

Si possono osservare di seguito le tabelle che mostrano i risultati ottenuti per ogni modello addestrato e le corrispettive loss e deviazioni standard.

In ogni tabella viene definito con *Valore* il numero di paper che in fase di test si sono classificati in una posizione compresa tra 0 e *Posizione* (escluso).

Come mostrato in tabella 4.1 (a) il modello addestrato sul dataset con i titoli di lunghezza maggiore di 20 caratteri, classifica il 68% delle istanze di test (272 su 400) tra le prime dieci posizioni, mentre 13% viene classificato oltre la centesima posizione. La loss finale ottenuta in questo modello è pari a 0.0059, mentre la deviazione standard è di 4.77.

Posizione	Valore
10	272
20	294
30	313
40	322
50	331
60	336
70	339
80	341
90	343
100	348
Loss	**DS**
0.0059	4.77

(a) Modello 20

Posizione	Valore
10	221
20	258
30	273
40	290
50	296
60	304
70	309
80	315
90	319
1000	323
Loss	**DS**
0.010	8.65

(b) Modello 30

Posizione	Valore
10	271
20	303
30	320
40	327
50	332
60	334
70	341
80	345
90	346
100	349
Loss	**DS**
0.0038	2.65

(c) Modello 40

Posizione	Valore
10	259
20	297
30	314
40	323
50	328
60	332
70	342
80	348
90	351
100	355
Loss	**DS**
0.0060	5.75

(d) Modello 60

Posizione	Valore
10	256
20	283
30	300
40	311
50	323
60	327
70	333
80	336
90	336
100	338
Loss	**DS**
0.0050	4.29

(e) Modello 70

Posizione	Valore
10	261
20	289
30	310
40	321
50	331
60	338
70	347
80	353
90	360
100	362
Loss	**DS**
0.0022	1.96

(f) Modello 80

Tabella 4.1: Risultati ottenuti dai diversi modelli in fase di test

Allo stesso modo possiamo osservare le tabelle relative agli altri modelli e notare che il modello peggiore sembra essere quello da 30 caratteri in figura 4.1 (b), mentre il migliore è quello con la soglia dei caratteri del titolo posta a 80, in figura 4.1 (f), con una loss pari a 0.0022 e una deviazione standard di 1,94.

Dai dati possiamo notare che nel complesso all'aumentare della soglia dei caratteri, la loss del modello tende a migliorare, anche se questo non si verifica esattamente per tutte le soglie per cui abbiamo testato il modello.

Si può osservare il grafico che rappresenta l'andamento dell'errore dei differenti modelli in figura 4.1

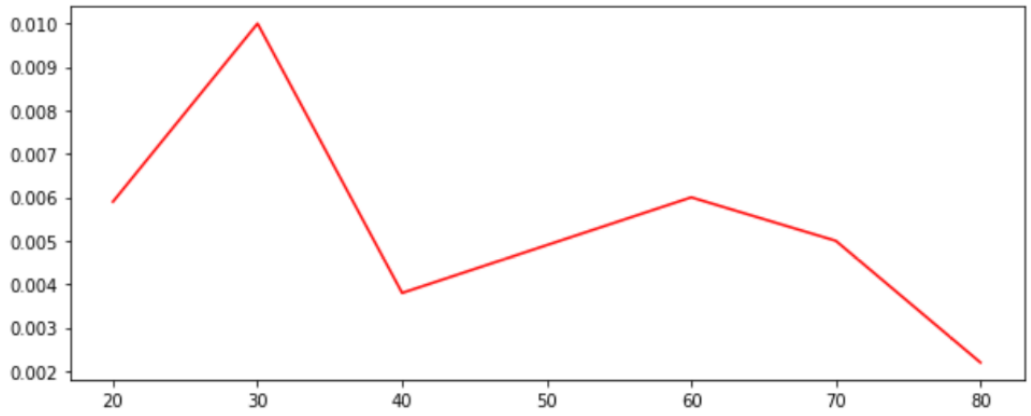

Figura 4.1: Andamento dell'errore nei modelli addestrati

4.4 Valutazione dei modelli sul dataset TREC-COVID e confronto con il modello CO-Search

In quest'ultima fase del progetto mi sono occupata di testare i modelli ottenuti sul dataset proposto da **TREC-COVID**, il quale consiste in una raccolta ridotta di documenti contenuti nel dataset originale di Kaggle sul COVID-19, creata al fine di valutare algoritmi di Information Retrieval relativi al virus.

TREC-COVID propone nel paper *"TREC-COVID: Constructing a Pandemic Information Retrieval Test Collection"*[**13**] una challenge di IR, per il supporto di ricercatori e medici durante questo periodo critico dovuto alla pandemia. I task sono suddivisi in più round, ognuno indipendenti l'uno dall'altro.

I modelli addestrati sono stati testati sul primo round, il quale consiste in 30 topic, composti da:

- query: una breve frase composta da parole chiave
- question: una domanda in linguaggio naturale
- narrative: una descrizione più informativa riguardante il topic

Si può osservare un esempio della composizione di un topic nella tabella 4.2.

query	coronavirus origin
question	what is the origin of COVID-19?
narrative	seeking range of information about the SARS-CoV-2 virus's origin, including its evolution, animal source, and first transmission into humans

Tabella 4.2: Esempio composizione di un topic TREC-COVID

Ogni topic è associato ad una lista di paper, per un totale di 8692 documenti, ognuno dei quali è annotato con la rilevanza relativa al determinato topic, in una scala da 0 a 2: 0 *non rilevante*, 1 *parzialmente rilevante*, 2 *rilevante*.

Ho quindi implementato il codice per testare i modelli sui 30 topic proposti dal primo round, ognuno su un dataset differente, contenente i documenti annotati per il rispettivo topic.

I sei modelli addestrati sono stati testati su tutte e tre le tipologie di input contenute nel topic, ottenendo così tre valutazioni per ogni modello.

Si sono infine confrontati i risultati finali con quelli ottenuti da **CO-Search**, il modello che ha ottenuto la performance migliore sulla challenge del primo round di TREC-COVID ed esposto nel paper *"CO-Search: COVID-19 Information Retrieval with Semantic Search, Question Answering, and Abstractive Summarization"* [**14**].

4.4.1 Metriche utilizzate per la valutazione finale

Per poter confrontare in modo esatto i nostri modelli con CO-Search si è deciso di valutarli con le stesse metriche utilizzati da quest'ultimo.

Precision (P)

La precision è una metrica standard già analizzata nel primo capitolo, utilizzata in questo caso solo sui primi N documenti restituiti dal modello, come indicato dalla convenzione standard $P@N$, mostrata nella seguente formula:

$$P@N = \frac{|relevant \quad documents \quad in \quad top \quad N|}{N}$$

In questo modo andiamo a calcolare la frequenza di documenti realmente rilevanti tra i primi N restituiti dal modello.

Normalized discounted cumulative gain (nDCG)

Questa metrica viene utilizzata per valutare la qualità del ranking restituito da un algoritmo di IR, in base al valore di rilevanza associato ai documenti restituiti e alla posizione che questi occupano nella classifica finale.

È ottima per valutare modelli che non utilizzano dati binari (rilevante - non rilevante), come in questo caso.

$$nDCG@N = \frac{1}{Q}\sum_{q=1}^{Q}\frac{DCG_p^{(q)}}{IDCG_p^{(q)}} \text{ , with } DCG_p^{(q)} = rel_1^{(q)} + \sum_{i=2}^{N}\frac{rel_i^{(q)}}{\log_2(i)}$$

dove $rel_i^{(q)}$ denota la rilevanza del documento i e $IDCG$ corrisponde al valore massimo di DCG che si può ottenere.

Mean average precision (MAP)

L'*avarage precision* (AP) è definita come l'integrale della curva di precision-recall sul dataset di una detemintata query. La MAP è la media delle AP calcolate su ogni query.

$$MAP = \frac{1}{Q} \sum_{q=1}^{Q} \int_{0}^{1} P_q(R)$$

dove R è la recall e P_q è la funzione di precision su R per una determinata query.

Lo scopo di questa metrica è simile a quello della $nDCG$, ma a differenza, di quest'ultima viene valutata su un dataset binario. Di conseguenza vengono considerati non rilevanti anche i documenti annotati in realtà parzialmente rilevanti, ottenendo una valutazione più scarsa rispetto a quella restituita dalla $nDCG$.

Binary preference (Bpref)

Questa metrica calcola quanto frequentemente i documenti rilevanti sono restituiti prima dei non rilevanti.

$$Bpref = \frac{1}{R} \sum_{r=1}^{R} 1 - \frac{|n \quad ranked \quad higher \quad than \quad r|}{R}$$

dove R sono il numero di documenti giudicati rilevanti, r è un documento rilevante restituito e n è un documento irrilevante restituito.

4.4.2 Risultati e confronto con il modello CO-Search

Come già anticipato ogni modello è stato testato per tre volte, calcolando la similarità che i documenti del dataset hanno con query, question e narrative.

I risultati finali mostrano come si ottenga una performance migliore utilizzando in input *question* rispetto a *query* e *narrative*, i quali restituiscono score abbastanza simili tra di loro.

Inoltre anche in questo caso il modello con soglia 80 caratteri sembra superare gli altri modelli nella maggior parte delle metriche utilizzate per valutarlo, in tutte e tre le tipologie di input.

In conclusione, il modello migliore tra quelli addestrati, per un problema di Information Retrieval su documenti riguardanti il COVID-19, è quello il cui training è stato fatto sui documenti i cui titoli hanno lunghezza minima di 80 caratteri. Inoltre i risultati migliori sono ottenuti dando in input al modello la domanda in linguaggio naturale.

In tabella 4.3 vengono raffigurati i risultati ottenuti dal modello da 80 caratteri per tutti e tre le tipologie di input.

	Query	**Question**	**Narrative**
P@5	0,4533	0,5399	0,4733
P@10	0,4033	0,5033	0,4266
nDCG@10	0,5134	0,6314	0,5310
MAP	0,2587	0,2981	0,2618
Bpref	0,2987	0,3593	0,3003

Tabella 4.3: Risultati ottenuti dal modello con soglia 80 caratteri

Osservando gli score ottenuti si potrebbe pensare che il modello non sia molto abile a classificare i documenti. Ad esempio la precision restituisce un valore pari circa a 0,5. Ciò vuol dire che nelle prime N posizioni solo il 50% dei paper sono rilevanti. Lo stesso risultato verrebbe restituito da un modello casuale nel caso le due classi *rilevante* e *non rilevante* fossero bilanciate.

Basta però chiarire che in realtà la classe *rilevante* si riferisca solo il 14% delle istanze in input per stabilire che il modello addestrato esegue un lavoro molto superiore a quello di un modello casuale.

Dopo aver determinato il modello migliore, ho infine confrontato i suoi risultati con quelli restituiti da CO-Search, i quali risultano essere i migliori ottenuti per lo specifico task.

Gli score ottenuti da CO-Search e osservabili in figura 4.4 sono ovviamente superiori a quelli ottenuti dal nostro modello.

P@5	$0,8267$
P@10	$0,7933$
nDCG@10	$0,7233$
MAP	$0,4870$
Bpref	$0,5176$

Tabella 4.4: Risultati ottenuti dal modello CO-Search

Confrontando i due modelli, si può notare come il nostro restituisca, rispetto alle altre metriche, un valore abbastanza alto nella nDCG. Ricordiamo che questo valore valuta la bontà del modello considerando anche il ranking dei paper giudicati parzialmente rilevanti e non trasformando quindi i documenti in un dataset binario (*rilevante - non rilevante*), come accade nel calcolo delle altre metriche utilizzate.

Di conseguenza uno score alto in questa metrica rispetto alle altre, può voler indicare che il nostro modello è abile nel classificare i documenti non rilevanti nelle posizioni finali, ma non altrettanto capace nel distinguere i documenti rilevanti da quelli parzialmente rilevanti.

Questo spiegherebbe anche il motivo per cui gli altri score siano così bassi e perché, ricalcolando le metriche considerando *rilevanti* anche i paper *parzialmente rilevanti*, si ottengano risultati molto più alti e vicini a quelli del modello CO-Search.

Un'ultima osservazione che vorrei mostrare è come il nostro modello classifichi 2 dei 30 topic testati con una precision sui primi 10 elementi (P@10), e di conseguenza anche sui primi 5 (P@5), con il punteggio massimo di 1. Quindi i primi 10 paper restituiti sono tutti considerati *rilevanti*, ossia etichettati con score 2. Si possono verificare i risultati di questi topic attraverso i seguenti screenshot in Figura 4.2 e 4.3.

```
Question: what kinds of complications related to COVID-19 are associated with diabetes
First 10 papers retrieved:
SCORE    TITLE
2 C-reactive protein levels in the early stage of COVID-19
2 Neutrophil-to-Lymphocyte Ratio Predicts Severe Illness Patients with 2019 Novel Coronavirus in the Early
 Stage
2 [Analysis of clinical features of 29 patients with 2019 novel coronavirus pneumonia].
2 Clinical and biochemical indexes from 2019-nCoV infected patients linked to viral loads and lung injury
2 A Tool to Early Predict Severe 2019-Novel Coronavirus Pneumonia (COVID-19) : A Multicenter Study using t
he Risk Nomogram in Wuhan and Guangdong, China
2 Proteomic and Metabolomic Characterization of COVID-19 Patient Sera
2 Risk assessment of progression to severe conditions for patients with COVID-19 pneumonia: a single-cente
r retrospective study
2 Clinical characteristics of 25 death cases infected with COVID-19 pneumonia: a retrospective review of m
edical records in a single medical center, Wuhan, China
2 A machine learning-based model for survival prediction in patients with severe COVID-19 infection
2 Early Prediction of Disease Progression in 2019 Novel Coronavirus Pneumonia Patients Outside Wuhan with
CT and Clinical Characteristics
```

Figura 4.2: Primi 10 documenti restituiti dal topic numero 24

```
Question: what is known about those infected with Covid-19 but are asymptomatic?
First 10 papers retrieved:
SCORE    TITLE
2 Of chloroquine and COVID-19
2 Role of Chloroquine and Hydroxychloroquine in the Treatment of COVID-19 Infection- A Systematic Literatu
re Review
2 Therapeutic Application of Chloroquine in Clinical Trials for COVID-19
2 Efficacy of hydroxychloroquine in patients with COVID-19: results of a randomized clinical trial
2 Combined prophylactic and therapeutic use maximizes hydroxychloroquine anti-SARS-CoV-2 effects in vitro
2 A systematic review on the efficacy and safety of chloroquine for the treatment of COVID-19
2 Chloroquine and hydroxychloroquine in the treatment of COVID-19 with or without diabetes: A systematic s
earch and a narrative review with a special reference to India and other developing countries
2 Quantifying treatment effects of hydroxychloroquine and azithromycin for COVID-19: a secondary analysis
of an open label non-randomized clinical trial (Gautret et al, 2020)
2 In Vitro Antiviral Activity and Projection of Optimized Dosing Design of Hydroxychloroquine for the Trea
tment of Severe Acute Respiratory Syndrome Coronavirus 2 (SARS-CoV-2)
2 A Rush to Judgment? Rapid Reporting and Dissemination of Results and Its Consequences Regarding the Use
of Hydroxychloroquine for COVID-19
```

Figura 4.3: Primi 10 documenti restituiti dal topic numero 27

Conclusioni

In questa tesi si sono andati ad analizzare gli approcci utilizzati per i problemi di Information Retrieval, dai primi metodi implementati fino ai più recenti, ponendo particolare interesse ai modelli Neurali che utilizzano tecniche di Machine Learning e Deep Learning.

Attraverso questa analisi si è voluta mostrare la rivoluzione intrapresa negli ultimi anni nel campo Natural Language Processing, grazie alle nuove tecniche Neurali e come questa sia tutt'ora in atto, mediante la continua creazione di modelli che raggiungono lo stato dell'arte, superando i precedenti.

Si è inoltre realizzato un progetto di Information Retrieval Neurale, su un task di recupero di documenti rilevanti riguardanti il COVID-19, sulla base di una domanda posta in linguaggio naturale.

Sono diverse le challenge di IR riguardanti il coronavirus, che sono state proposte negli ultimi mesi. Ciò è dovuto principalmente alla continua crescita del numero di documenti pubblicati inerenti al nuovo virus e all'impossibilità di consultare un numero così elevato di paper, senza l'aiuto di un algoritmo capace di selezionare i documenti di reale interesse.

Si è utilizzato come modello di base BERT (la versione SciBERT nello specifico), il quale nel 2018, anno di pubblicazione, rappresentava lo stato dell'arte su molti task NLP.

A distanza di soli due anni, sono diversi i nuovi modelli proposti che sono stati capaci di raggiungere o addirittura superare le performance di BERT, confermando il continuo progresso che è in atto tuttora in questo ambito.

Bibliografia

[1] Vannevar Bush, "As We May Think", 1945.

[2] MIT Alexander Amini, MIT Introduction to Deep Learning , 2020 URL: https://www.youtube.com/watch?v=njKP3FqW3Sk

[3] MIT Ava Soleimany, Recurrent Neural Networks, URL: https://www.youtube.com/watch?v=SEnXr6v2ifU

[4] Bhaskar Mitra, Nick Craswell (Microsoft), "Neural Models for Information Retrieval", 2017.

[5] Jacob Devlin et al. (Google), " BERT:Pre-training of Deep Bidirectional Transformers for Language Understanding", 2019.

[6] Aurélien Géron, "Hands-on Machine Learning with Scikit-Lear, Keras and TensorFlow", O'Reilly Media, 2019.

[7] Jay Alammar, The Illustrated Transformer, 2018, URL: http://jalammar.github.io/illustrated-transformer

[8] Ashish Vaswani et al. (Google) "Attention Is All You Need", 2017

[9] Amirhossein Zazemnejad, Transformer Architecture: The Positional Encoding, 2019, URL: https://kazemnejad.com/blog/transformer_architecture_positional_encoding

[10] Arman Cohan et al. (Allen Institute for Artificial Intelligence), "SPECTER: Document-level Representation Learning using Citation-informed Transformers", 2020.

[11] Iz Beltagy et al. (Allen Institute for Artificial Intelligence), "SCIBERT: A Pretrained Language Model for Scientific Text", 2019.

[12] Kaggle Competition, "COVID-19 Open Research Dataset Challenge(CORD-19)", 2020, URL: https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge

[13] Ellen Voorhees et al., "TREC-COVID: Constructing a Pandemic Information Retrieval Test Collection", 2020.

[14] Andre Esteva et al., "CO-Search: COVID-19 Information Retrieval with Semantic Search, Question Answering, and Abstractive Summarization", 2020.