ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

GENERARE CON TRANSFORMER LINEARE SPIEGAZIONI DI CONCEPT SET MEDICALI COSTITUITI DA TERMINI CORRELATI ESTRATTI DA SOCIAL POST DI PAZIENTI

Elaborato in
Text Mining

Relatore
Prof. Gianluca Moro

Co-relatore
Dott. Ing. Giacomo Frisoni

Presentata da
Alessia Ventani

Terza Sessione di Laurea
Anno Accademico 2019 – 2020

PAROLE CHIAVE

Linear Attention

Conditional Language Model

Natural Language Generation

FLAX

Rare diseases

Introduzione

Negli ultimi anni si è assistito alla diffusione di modelli di reti neurali in ambito Natural Language Processing (NLP). I recenti traguardi sulla generazione e comprensione automatica del linguaggio naturale hanno determinato una profonda popolarità di tali soluzioni anche in contesti al di fuori della ricerca. Si pensi per esempio a GPT-3 di OpenAI, capace di predire testo con una qualità sintattica e semantica tale da risultare spesso indistinguibile da quello prodotto da un umano, generando preoccupazioni anche sul fronte etico e impedendone la pubblicazione. La caratteristica principale di questi modelli, basati sull'architettura Transformer, è l'elevata dimensione, intesa come numero di pesi di cui si compongono. Sebbene ciò abbia consentito il raggiungimento di risultati sempre più accurati e sorprendenti, i ricercatori si domandano se sia questa la strada per la risoluzione delle problematiche ancora aperte e se ciò costituisca un importante ostacolo alla democratizzazione della ricerca. Le risorse computazionali normalmente accessibili ai gruppi di ricerca, infatti, sono sempre più insufficienti per l'addestramento dei modelli sopracitati. I costi richiesti sono sostenibili solo da grandi multinazionali quali Google, Facebook e OpenAI, acquisendo di fatto un dannoso monopolio sul settore. Con l'obiettivo di ridurre il numero di risorse richieste per l'addestramento dei modelli e abilitare l'elaborazione di sequenze più lunghe, si è sviluppata una nuova area di ricerca incentrata sull'individuazione di soluzioni atte a ridurre il costo di Attention da quadratico a lineare, rispetto alla lunghezza dell'input. Tali contributi costituiscono un passo importante nella direzione della democratizzazione della ricerca, e al tempo stesso rendono possibile il perseguimento di task prima non affrontabili. Centri universitari e gruppi di ridotte dimensioni possono così applicare, estendere e migliorare lavori altrimenti non accessibili.

In questo contesto si posiziona il lavoro della tesi. Considerando come punto di partenza la recente architettura del Performer [8], introdotta da Google Research, ci si pone l'obbiettivo di replicare i risultati descritti nella pubblicazione e applicare FAVOR+ a un caso di studio medico. In particolare ci si concentra sulla Acalasia esofagea, malattia rara (MR) con una incidenza stimata di 1/10000. I pazienti MR tendono a cadere in uno sconforto dovuto anche alla mancanza di informazioni sul proprio disturbo. Per questo motivo,

nel corso degli anni, sono nate comunità virtuali sui social che raggruppano pazienti e specialisti con il fine ultimo di aiutarsi e condividere le proprie esperienze. L'analisi automatica delle informazioni presenti nei post pubblicati (i.e., la voce dei pazienti) permette di apprendere una base di conoscenza utile per un paziente o un sanitario, e contenente informazioni preziose per l'avanzamento della ricerca stessa. In questo si riprende [12] una metodologia di text mining descrittivo di recente pubblicazione e applicata al medesimo contesto presentato, con cui gli autori sono stati in grado di estrarre in modo non supervisionato correlazioni medicali quantificate a livello statistico con un punteggio F1 del 79% e spiegazioni a fenomeni di interesse, intese come insiemi di parole altamente correlate col fenomeno stesso.

Secondo obiettivo di questa tesi è pertanto la generazione di frasi (grammaticalmente corrette, di senso compiuto, contenenti informazioni fattuali, e a elevato contenuto informativo) capaci di fornire una spiegazione testuale alle correlazioni tra n-uple di termini estratte con la suddetta metodologia. Questo task corrisponde alla generazione di frasi di senso compiuto che mettano in correlazione delle parole date input e che aggiungano conoscenza realizzando una vera e propria spiegazione della correlazione degli input.

Questo aspetto è di fondamentale importanza poiché presenta una complessità notevole: non è presente una trattazione estesa in letteratura (dove tendenzialmente ci si limita alla predizione di maschere a livello di parola o a porzioni di testo) e infine ha innumerevoli applicazioni. Oltre a quello è già indicato e su cui ci si è concentrati si riportano:

- generazione di recensioni artificiali in ambito e-commerce che racchiudano l'esperienza descritta da diversi utenti in forma aggregata;
- l'ampliamento di training set di frasi etichettate con relativi eventi, generando frasi sintetiche di senso compiuto a partire dalla notazione usata per la rappresentazione degli eventi stessi;
- la realizzazione di sistemi che permettano di aiutare le persone affette da afonia di esprimersi in modo più semplice;
- aiutare un utente nell'apprendimento di una nuova lingua;
- in generale riassumere, ampliare e spiegare il punto di vista di persone in maniera automatica.

L'elaborato, dunque, prevede una trattazione suddivisa in sei capitoli che trattano i seguenti argomenti.

- **Capitolo 1 - I Transformer**. Un'illustrazione delle scoperte tecnologiche che hanno portato alla realizzazione di questa architettura con una

spiegazione degli aspetti tecnici che hanno influito sulla suo successo. In questo capitolo si descrivono anche i due modelli utilizzati per il lavoro, BERT e T5, le loro caratteristiche e le loro differenze.

- **Capitolo 2 - I Transformer efficienti**. Una trattazione dei limiti dell'architettura base del Trasformer e delle principali tecniche che sono state introdotte per migliorale. In particolare si è descritto il recente modello del Performer e le caratteristiche del meccanismo del FAVOR+ che sono stati utilizzati negli addestramenti.

- **Capitolo 3 - Modelli e tecnologie**. Un excursus sulle tecnologie e i framework utilizzati per il lavoro. L'attenzione è stata posta sulla nuova libreria FLAX e sulle implementazione di BERT e T5 in questa.

- **Capitolo 4 - Il contesto del problema**. Un breve capitolo sull'importanza della ricerca medica in questo ambito, sul lavoro già effettuato e sul dominio applicativo considerato con il dataset realizzato.

- **Capitolo 5 - I contributi**. Si descrivono passo a passo i contributi effettivi realizzati e le modifiche implementate al codice di partenza al fine di applicare il Performer ai modelli voluti. Dalla modifica dei modelli all'implementazione sui server per l'addestramento.

- **Capitolo 6 - I risultati raggiunti**. Si riportano i risultati raggiunti sia in performance sia nelle predizione fatte con i modelli considerati e si effettua un confronto fra i risultati attesi e quelli trovati.

Indice

Elenco delle figure

Capitolo 1

Il Transformer

Il *Transformer* rappresenta un modello di rete neurale che ha rivoluzionato il settore del machine learning e ha permesso il raggiungimento di innumerevoli traguardi in diversi task, rappresentando tutt'oggi lo stato dell'arte in numerosi di questi.
Per comprendere la struttura del Transformer e le innovazioni che esso ha portato, è necessario analizzare le scoperte e i lavori che hanno contribuito alla costruzione di questo modello.

1.1 Il modello sequence-to-sequence

I *sequence-to-sequence* sono modelli di deep-learning che hanno raggiunto un considerevole successo in ambiti come la traduzione di testi, realizzazione di riassunti e composizione di didascalie per immagini. Essi, anche chiamati seq2seq, vengono illustrati per la prima volta in una pubblicazione realizzata da un gruppo di ricerca di Google nel 2014 [28] e da un'altra di un gruppo di ricercatori delle università di Montreal e Le Mans [7]. Dal 2016 [32], inoltre, queste reti sono utilizzate da Google Translate, famoso tool per la traduzioni di frasi e paragrafi.

Al fine di analizzarne il funzionamento, si descrive di seguito l'applicazione di questo modello nel task per il quale è stato ideato ovvero la traduzione di testi in altre lingue. Una rete seq2seq prende in input una serie di input (parole, lettere, feature di una immagine ecc) e ha come output un'altra sequenza di item. Nel task preso in considerazione, come è facile intuire, la sequenza di input è composta da parole e l'output da una nuova sequenza di esse che corrispondono alle precedenti tradotte nella lingua voluta.

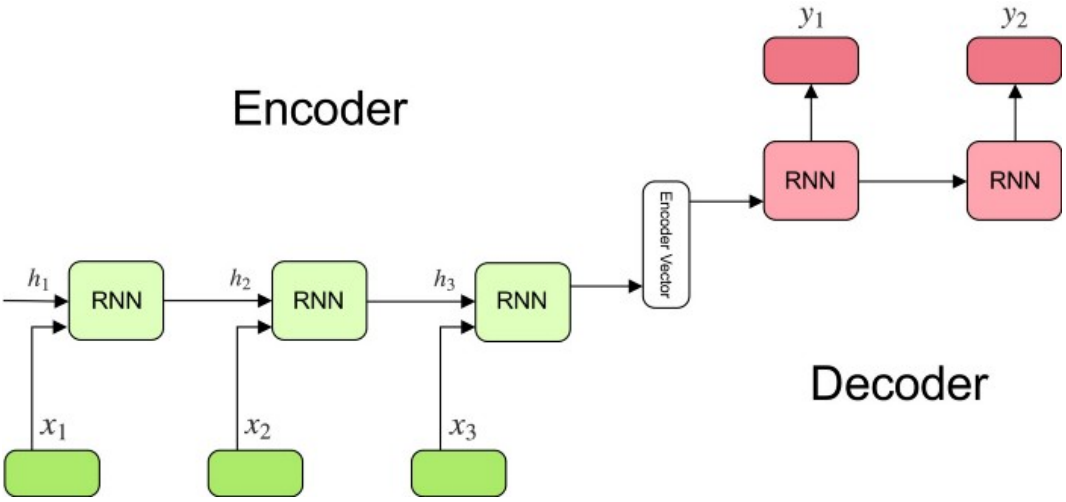


Figura 1.1: Architettura del modello seq2seq [16]

Analizzando la Figura 1.1, è possibile individuare nell'architettura di questo modello due componenti principali.

- *Encoder*. Costituisce la prima componente e processa la sequenza di input raccogliendo informazioni in un vettore, tipicamente di numeri, che prende il nome di contesto, in Figura 1.1 denominato *encoder vector*. Ogni item di input viene processato singolarmente, aggiornando di passo in passo l'*hidden state* della rete con le nuove informazioni acquisite, procedendo poi con l'elemento successivo. L'ultimo hidden-state corrisponde al contesto.

- *Decoder*. Effettua il processo inverso. Ricevendo il contesto dall'encoder e considerandolo come primo hidden state, lo elabora per produrre come output una sequenza di parole. Nel task di traduzione la nuova sequenza corrisponde alla frase di input tradotta nella lingua voluta.

In queste reti ogni input di lunghezza variabile viene rappresentato con un vettore di lunghezza predefinita e fissa. Questa caratteristica può portare al problema del *vanishing gradient* e per evitare quest'ultimo, il docoder e l'encoder sono costituiti tipicamente, nel modello originale, da reti neurali ricorrenti LSTM [14] o GRU [7].

Come detto precedentemente, ogni parola viene rappresentata da un vettore. Questa conversione avviene attraverso un algoritmo di *word embedding*. L'obbiettivo è quello di trasferire le parole in un nuovo spazio capace di riflettere la loro correlazione o similarità semantica attraverso la vicinanza tra i vettori che le rappresentano. Questo processo permette di catturare una elevata quantità di informazione e può essere effettuato con reti pre-addestrate o con modelli

che vengono addestrati assieme all'intera rete. In entrambi i casi è importante definire la lunghezza dell'embedding desiderato: i valori più utilizzati sono 256, 512 e 1024.

Occorre tener presente che, nella versione presentata nel 2014, i modelli seq2seq presentano diverse limitazioni.

- Un collo di bottiglia informativo:
  - le sequenze di lunghezza elevata non vengono processate correttamente, la dimensione fissa dei vettori impedisce al modello di rappresentare input di lunghezza elevata correttamente e apprendere per intero le informazioni contenute in esse;
  - maggiore è la dimensione della sequenza minori sono le prestazioni della rete (anche LSTM e GRU);
  - il vettore di contesto di fatto limita il potere di predizione del decoder e rappresenta un collo di bottiglia per l'informazione trasmessa fra i due componenti.
- Asimmetria nel processare la sequenza di input: viene data più importanza agli ultimi step della sequenza mentre il contesto dei primi item ha un effetto molto limitato sul risultato finale.
- Limitazione nelle performance: la natura sequenziale delle reti neurali ricorrenti impedisce la parallelizzazione del calcolo del risultato e occorre che tutte le computazioni siano terminate prima di poter calcolare l'output. Questa condizione ha come conseguenza una limitazione nelle performance e un aumento dei tempi di calcolo.

Per queste ragioni sono state introdotte alcune ottimizzazioni al modello originale.

- Attention [34]: meccanismo che permette al decoder di concentrare la propria attenzione su determinati item dell'input contenenti le principali informazioni della frase.
- Beam Search [36]: la parola o le parole con più probabilità vengono selezionate come output del decoder. Nel caso di più parole il risultato è rappresentato da un albero che illustra le principali soluzioni.
- Bucketing: la lunghezza fissa delle reti RNN è resa possibile dall'utilizzo del padding, ovvero l'aggiunta di uno specifico token per le frasi che non raggiungano la lunghezza voluta. Questo, se si hanno input di lunghezza

molto diverse, costituisce uno spreco di spazio. La tecnica del bucketing permette di raggruppare le frasi in base a dei range di lunghezza e, in questo modo, quelle di lunghezza simile verranno processate assieme da reti con lunghezza massima diversa, limitando così lo spreco di memoria.

Nella Sezione 1.2 viene descritta in dettaglio la prima ottimizzazione elencata e i suoi successivi sviluppi che hanno portato alla realizzazione del Transformer.

1.2 Il meccanismo dell'Attention

1.2.1 Il word alignment

Come discusso in sezione 1.1, riassumere tutte le informazioni per una buona predizione in un unico vettore può risultare complesso e, per lunghe sequenze di parole, quasi impossibile. Una soluzione potrebbe essere quella di mantenere un vettore di informazioni distinte per ogni parola; tuttavia ciò determina elevati tempi di computazione e un significativo consumo di memoria mantenendo informazioni su elementi non importanti della frase che potrebbero essere scartate. Un approccio di questo genere può funzionare solo se il modello possiede la capacità di "selezionare" e "porre la propria attenzione" sulle parole più importanti e dal maggior valore informativo.

Il modello di rete neurale deve quindi essere in grado di concentrarsi sugli input più importanti così da poter scegliere il giusto output da predire. Risulta necessario allineare le parole di input con quelle di output. Il concetto di *word alignment* non è nuovo ma è ampiamente utilizzato in task di traduzione e word sense discovery. A questo scopo risulta fondamentale identificare le relazioni fra le parole della frase per produrre predizioni accurate anche con parole non ordine o che, per esempio, non hanno una traduzione precisa.

1.2.2 Dal word alignment all'Attention

Per allineare le parole correttamente è necessario introdurre un livello che aiuti il decoder a comprendere quali input sono più importanti per ogni previsione. Questo livello prende il nome di Attention e assegna, attraverso il calcolo di funzioni matematiche che verranno analizzate nelle sezioni successive, dei pesi diversi ad ogni input per ogni predizione. Il decoder, quando deve individuare la predizione, viene fortemente influenzato dagli hidden state degli input con peso maggiore rispetto ai restanti. Lo scopo dell'Attention e dell'allineamento è, dunque, quello di calcolare per ogni parola un vettore di score.

Una versione semplificata del calcolo dell'attention può essere composta dai seguenti passi.

1. Dati tutti gli hidden state disponibili per l'encoder e per il decoder, si ottengono i pesi calcolando i prodotti scalari fra ogni encoder state e i decoder hidden state. Se uno di questi è maggiore degli altri allora la prossima previsione è influenzata maggiormente da esso rispetto agli altri.

2. Il risultato dell'operazione precedente viene passato attraverso la funzione di softmax che trasforma i valori dei pesi in probabilità, valori compresi fra 0 e 1, che costituiscono la distribuzione dell'attention.

3. Ogni hidden state dell'encoder viene moltiplicato per il peso dell'attention passato attraverso la softmax, ottenendo il vettore di allineamento.

4. sommando i vali ottenuti si ottiene il vettore di contesto.

È importante sottolineare come questo meccanismo consenta al decoder di ottenere delle informazioni riguardanti tutti gli hidden state dell'encoder ma pesati per la loro importanza nella previsione del prossimo output.

1.2.3 Gli elementi dell'Attention

Il calcolo dell'Attention è basato su tre elementi fondamentali: *Key*, *Value*, *Query*. Per comprendere il ruolo di questi ultimi è utile considerare i seguenti esempi noti, Standford [1].

Le chiavi perse. Un ragazzo deve uscire di casa ma non trova più le sue chiavi. Davanti ha una scelta da compiere: o passare diverso tempo a cercarle setacciando ogni stanza della casa in maniera casuale e ritardare la sua uscita o invocare l'aiuto provvidenziale della madre che, con molto probabilità, riuscirà a trovarle prima di lui. La madre, con un approccio più pragmatico del figlio, penserà al posto più probabile dove si possono trovare le chiavi, lo comunicherà al figlio e probabilmente avrà indovinato il posto in cui sono state lasciate le chiavi. Il meccanismo dell'Attention effettua la stessa operazione della madre: considerata una richiesta, seleziona il posto dove più verosimilmente ha senso cercare la chiave per poi trovarla. Dunque l'Attention accoppia una Key e una Query assegnando un Value al posto dove è più probabile che si trovi la Key.

La ricerca su YouTube. Quando si scrive una ricerca per un video di YouTube, il motore di ricerca mappa la richiesta, query, su un set di chiavi, key

[1] https://www.coursera.org/learn/attention-models-in-nlp/home/welcome

(per esempio il titolo del video, la descrizione...) associate ai video candidati nel database per poi mostrare i video che meglio soddisfano la richiesta che corrispondono ai value.

Con questi esempi è facile intuire a cosa corrispondono gli elementi dell'Attention: di fronte una richiesta, che nel modello rappresenta la Query, si associa ad essa una Key e si assegnano alle varie posizioni un Value, il più alto fra questi indica il punto dove è più semplice e probabile trovare la corrispondenza con la richiesta fatta.

1.2.4 La Dot-Product Attention

Nelle sezioni precedenti sono stati illustrati i concetti fondamentali e il meccanismo generale dell'Attention. Di seguito, invece, si illustra nel dettaglio le formule del tipo di Attention più diffusa: la Dot-Product Attention.
Come in precedenza, per descriverla al meglio, si considera il task di traduzione dall'inglese alla lingua tedesca. In questo contesto:

- i word-embedding in inglese rappresentano le key e i value, come nella maggior parte dei task i due coincidono;
- i word-embedding in tedesco rappresentano le query e le key del decoder;
- i value sono coppie che vengono dagli hidden state dell'encoder mentre le query provengono dagli hidden state del decoder.

L'obbiettivo dell'Attention, in questo task, è, partendo da una parola tedesca, cercare fra le frasi inglesi e trovare la posizione di input con le parole più vicine. Per raggiungere questo scopo si eseguono i passaggi illustrati di seguito. Come prima operazione, all'interno dell'Attention layer si calcola il prodotto scalare, appunto dot-product, fra le query e le key. Questa operazione risulta essere fondamentale poiché rappresenta una misura di similarità fra vettori: i valori che si ottengono determinano quanto occorre concentrarsi su un determinato input in inglese al fine di predire una parola in tedesco.
Successivamente si effettua il calcolo della softmax e, con questo passaggio, gli score che si ottengono diventano positivi e in un range di valore da zero a uno, costituendo di fatto una probabilità.
Infine, si moltiplicano le probabilità con i value: la sequenza di output risulta essere così pesata indicando quanto ogni parola, o combinazione di parole, sarà influente nella predizione della prossima parola tedesca. L'intuizione è quella di mantenere stabile il valore delle componenti che interessano e diminuire quello delle rimanenti moltiplicandole per un valore minore di uno.
Questo meccanismo è flessibile e permette di collegare parole che, da una lingua

all'altra, possono avere posizioni diverse all'interno della frase e gestisce le differenti grammatiche. L'intero processo è riassunto in Figura 1.2.

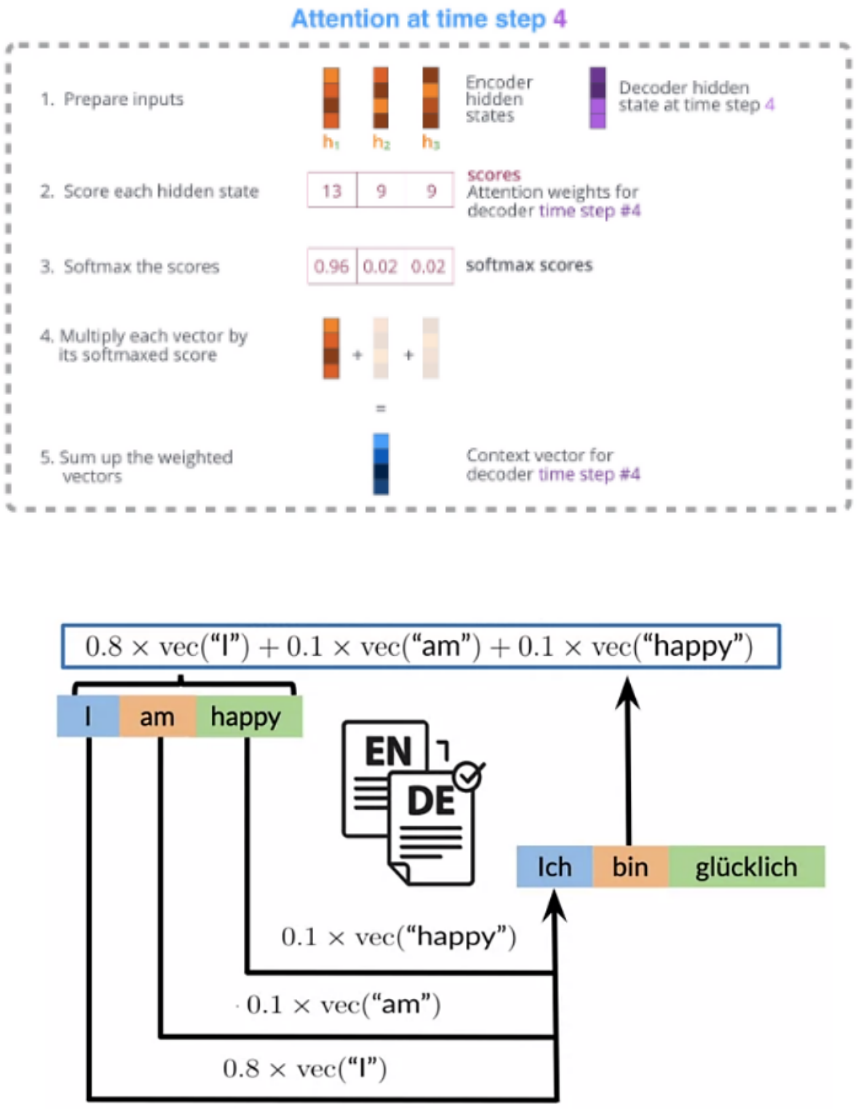


Figura 1.2: I passaggi principali del calcolo dell'Attention [2]

L'intero insieme delle operazioni descritte precedentemente può essere scritto in forma matriciale e riassunto nella seguente formula:

$$Attention(Q, K, V) = softmax(QK^T)V$$

dove:
$Query = Q : [L_Q, D_K]$

$Keys = K : [L_K, D_k]$
$Values = V : [L_V, D_v]$
L = lunghezza di query, key e value
D = dimensione arbitraria di query, key e value

Con questa formulazione è possibile ottenere l'Attention attraverso il calcolo di due moltiplicazioni tra matrici e l'applicazione della funzione di softmax. Inoltre è possibile sfruttare l'esteso supporto a livello di framework per quanto concerne l'uso di acceleratori hardware (come GPU e TPU), capaci di velocizzare significativamente e parallelizzare l'esecuzione di prodotti matriciali.

La formula descritta precedentemente però presenta un problema: la somma di prodotti scalari tende a aumentare enormemente in base alla dimensione delle query e delle key, cambiando la distribuzione del risultato e sbilanciandolo versi valori vicini allo zero o vicini all'uno. Per questa ragione esistono diverse normalizzazioni che possono essere applicate alla funzione di base. Con la Scaled Dot-Product Attention inoltre si possono sfruttare le implementazioni hardware, che sono supportate praticamente ovunque, per il calcolo veloce di moltiplicazioni tra matrici. La nuova formula dell'Attention prevede che il prodotto fra matrici sia diviso per la radice quadrata della lunghezza delle query e delle key:

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

Questa normalizzazione previene che il gradiente della funzione risulti essere estremamente piccolo o estremamente grande rendendolo più stabile e migliorando le performance del modello. Questa variante prende il nome di *Scaled Dot-Product Attention* il cui schema è riassunto in Figura 1.3

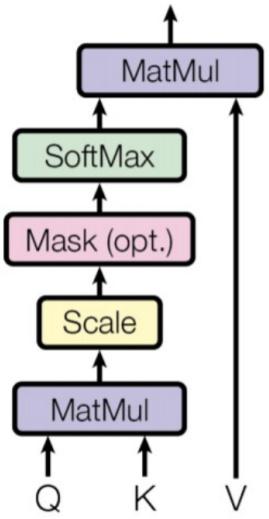


Figura 1.3: Scaled Dot Product Attention [34]

1.2.5 Il calcolo delle matrici

È stato descritto precedentemente come i componenti fondamentali del modello siano le Key, le Query e i Value. Di seguito si analizza come queste tre matrici vengono generate dall'input iniziale.

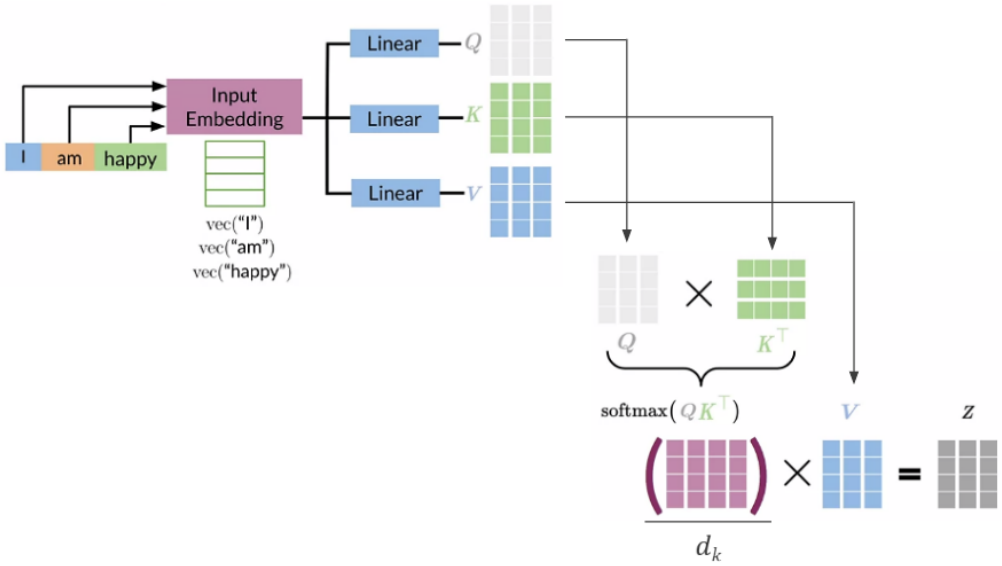


Figura 1.4: Le matrici dell'Attention [2]

Come è possibile osservare in Figura 1.4, i valori dei vettori q, k e v per un singolo input si ottengono alimentando tre distinti livelli lineari con esso. Questo processo si ripete per ogni input, che sia una parola (N-gram word level) o una sua parte (N-gram subword level, e.g., WordPieces), ottenendo così le tre matrici distinte Q, K e W. Per calcolare questi valori dunque si deve effettuare una moltiplicazione con le matrici $W_Q W_K$ e W_V, i cui valori devono essere addestrati con il resto della rete.
In generale i nuovi valori ottenuti hanno dimensione minore rispetto a quella del vettore di embedding iniziale ma questo rappresenta una scelta architetturale da effettuare. Inoltre, per velocizzare il calcolo, si è diffusa la tecnica di MultiHead-Attention al fine di parallelizzare e ridurre i tempi di computazione; questa ottimizzazione verrà descritta in Sezione 1.2.6.

1.2.6 Le tipologie di Attention

Nei paragrafi precedenti è stato analizzato il meccanismo generico dell'Attention, ma nel tempo sono state introdotte diverse varianti rispetto alla versione originale. Di seguito verranno illustrate le tre principali.

- *Encoder-Decoder Attention.* Una frase del decoder considera Q e K provenienti da differenti frasi. È il meccanismo visto nel precedente esempio di traduzione dall'inglese al tedesco: da "I" si ottiene "Je"/"Ich"/"Yo".

- *Casual Self Attention.* Utilizzata per la generazione di testo, dove in una singola frase le parole sono predette solo in funzione delle precedenti. In questa variante Q e K provengono dalla stessa frase e per questo prende il nome di Self-Attention. Il punto cruciale è che le parole possono considerare le parole precedenti ma non le successive o future. Dal punto di vista del calcolo, i valori di QK^T vengono sommati ad una matrice costante triangolare chiamata mask che presenta valori posti a - per le parole future permettendo così di non considerarle. Questo meccanismo può essere considerato come un caso particolare della Dot-Product Attention, Figura 1.5 [2].

[2] https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

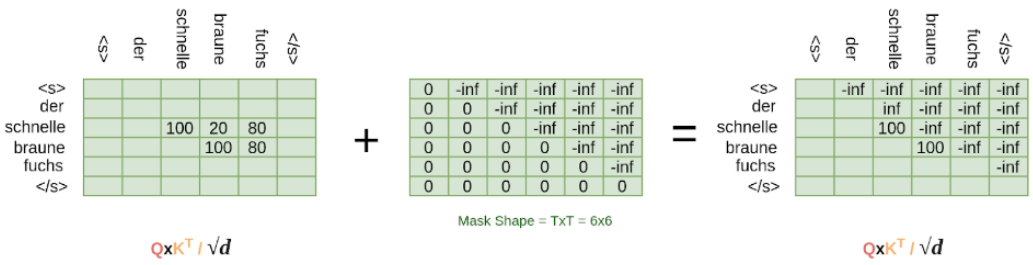


Figura 1.5: Casual Self Attention

- *Bi-Directional Self Attention*: le parole possono considerare sia le parole precedenti che quelle future. Questa Attention è utilizzata nei modelli presi in esame in questa tesi, quali BERT e T5. Figura 1.6.

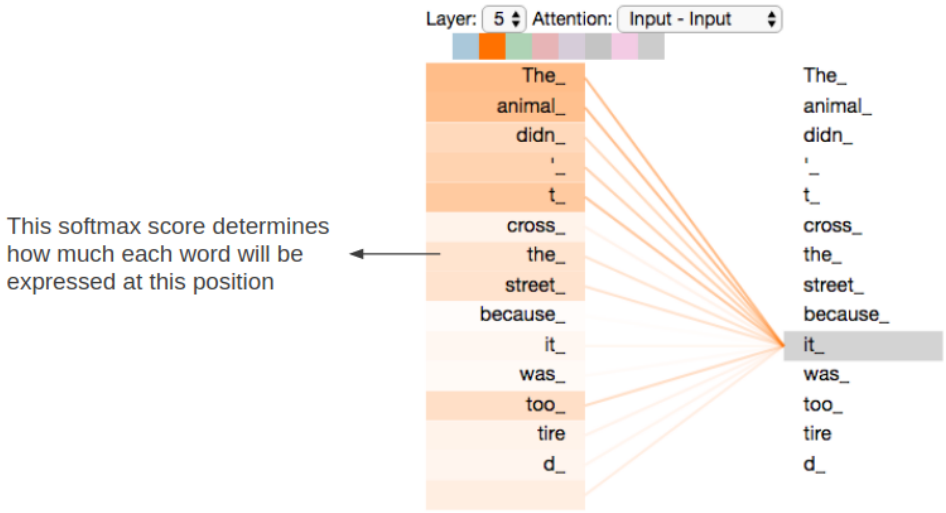


Figura 1.6: Bi-Directional Self Attention [2]

1.2.7 Un esempio di calcolo

Si conclude questa analisi dell'Attention riportando un esempio di calcolo di quest'ultima [5].

In particolare, si consideri il vettore di input $['Hello','\;','\,how','\,are','\,you','\,?']$ e si supponga che il risultato dell'operazione di *Attention* per il *token* "Hello" sia $[0.1, 0, 0.06, 0.1, 0.6, 0.14]$ come visibile in Figura 1.7.

$$
\begin{array}{c c}
 & \begin{array}{cccccc} Hello & , & how & are & you & ? \end{array} \\
\begin{array}{c} Hello \\ , \\ how \\ are \\ you \\ ? \end{array} &
\begin{pmatrix}
0.1 & 0 & 0.06 & 0.1 & 0.6 & 0.14 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots
\end{pmatrix}
\end{array}
$$

Figura 1.7: Risultato del prodotto $softmax(Q_i K_i^T)$ [5]

Eseguendo un successivo prodotto con il vettore d_v si ottiene il risultato di Figura 1.8. Al termine delle operazioni sopra indicate, la parola "Hello" può essere interpretata come una combinazione pesata di vettori di V_i sui *token* di input. Con il meccanismo di *Multi-Head Attention*, durante il corso dell'addestramento, esso diventa la concatenazione di h combinazioni di elementi addestrati su diverse proiezioni, rappresentando così la relazione fra il *token* selezionato e il vettore di *input*, come mostrato in Figura 1.8.

$$
\begin{array}{c c c}
 & \begin{array}{cccccc} Hello & , & how & are & you & ? \end{array} & d_v \\
\begin{array}{c} Hello \\ , \\ how \\ are \\ you \\ ? \end{array} &
\begin{pmatrix}
0.1 & 0 & 0.06 & 0.1 & 0.6 & 0.14 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots
\end{pmatrix} &
\begin{pmatrix} v_{Hello} \\ v_{,} \\ v_{how} \\ v_{are} \\ v_{you} \\ v_{?} \end{pmatrix} =
\end{array}
$$

$$
\begin{array}{c c}
 & < - - - - - - - - - - - - d_v - - - - - - - - - - - - > \\
\begin{array}{c} Hello \\ , \\ how \\ are \\ you \\ ? \end{array} &
\begin{pmatrix}
0.1v_{Hello} + 0v_{,} + 0.06v_{how} + 0.1v_{are} + 0.6v_{you} + 0.14v_{?} \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots
\end{pmatrix}
\end{array}
$$

Figura 1.8: Risultato dell'*Attention* [5]

1.3 L'architettura del Transformer

Nel Giugno del 2017, un gruppo di ricercatori di Google, guidato da Ashis Vaswani, ha pubblicato un paper dal titolo *Attention is all you need* [34] nel quale viene descritta l'architettura chiamata *Transformer* basata sul meccanismo dell'*Attention*.

1.3.1 Le caratteristiche principali

Il Transformer è uno dei modelli di deep learning più versatili: Similmente alle RNN, la sua architettura è pensata per poter lavorare in maniera efficiente con dati sequenziali e in generale con qualsiasi tipo di task sequenziale. Oggi giorno è utilizzata in moltissimi task di elaborazione del testo come machine translation, text summaritazion, autocompletion, Named Entity Recognition and linking, question answering, sentimental analysis, classification e altri ancora.

La sua struttura, con qualche variante, è alla base di innumerevoli reti neurali che negli ultimi anni si sono susseguite rapidamente nel tempo, divenendo lo stato dell'arte su numerosi benchmark e consentendo il raggiungimento di traguardi in aree di ricerca come NLP/NLU prima ritenuti lontani o impossibili. Tra questi ci sono anche i modelli che sono stati utilizzati in questo lavoro: BERT [9] e T5 [24].

Alla base del successo di tale architettura, vi sono i seguenti vantaggi.

- permette il parallelismo di calcolo. Il Transformer non è un modello sequenziale e dunque molto più semplice da parallelizzare e da accelerare. In particolare, l'Attention viene utilizzata per elaborare dati sequenziali in parallelo,richiedendo a livello di calcolo un singolo step per ogni passo di addestramento.

- Minore perdita di informazione. Nel Transformer si attenua significativamente il problema della perdita di informazione nell'elaborazione di sequenze lunghe tipica delle reti RNN. A differenza delle RNN, il numero di gradienti da calcolare non è pari alla lunghezza dell'input, ma a uno soltanto. Infine, con questo meccanismo, il Transformer non soffre del problema del vanishing gradient correlato alla lunghezza dell'input.

Di seguito si illustrano i componenti principali dell'architettura.

1.3.2 L'encoding posizionale

Una caratteristica del Transformer, è l'assenza sia della ricorrenza che della convoluzione. Questa porta alla necessità di aggiungere dell'informazione

riguardante le posizioni assolute o relative degli input all'interno della frase. Per raggiungere questo scopo si utilizza il *positional encoding*, un vettore sommato all'embedding di un token, rappresentante l'informazione spaziale e avente la stessa dimensione dell'input. Per calcolare i valori nel vettore di positional encoding si possono utilizzare diverse funzioni, sia addestrabili che fisse. Nell'architettura Transformer originaria *Transformer* si utilizza una funzione definita come segue che va a comporre la matrice P:

$$P_{p,2i} = sin(\frac{p}{1000\frac{2i}{d}}) \text{ e } P_{p,2i+1} = cos(\frac{p}{1000\frac{2i}{d}})$$

In questa espressione $P_{p,i}$ rappresenta l'i-esimo componente dell'*embedding* per la parola collocata alla alla p-esima posizione della frase. Con questa formulazione si riesce a registrare sia la posizione assoluta dell'*embedding* che la sua posizione relativa inoltre, questa funzione può essere applicata a frasi di lunghezza arbitraria.

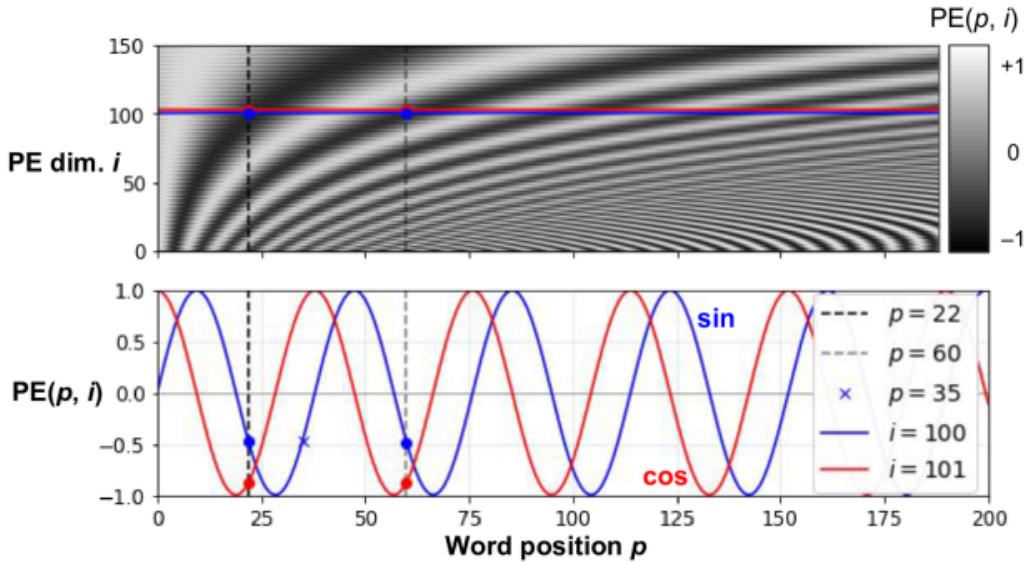


Figura 1.9: Matrice del *positional embedding* del seno e coseno (trasposta, sopra), con il dettaglio di due valori di i (sotto) [13]

Come si può osservare in Figura 1.9, per ogni posizione di ottiene un *positional embedding* univoco [13].

1.3.3 La Multi-Head Attention

Nel Transformer è stato introdotto un meccanismo aggiuntivo alla semplice Attention chiamato Multi-Head Attention. Esso sostituisce il layer ricorrente

delle seq2seq ed emula la ricorrenza delle sequenze. Il principio di base è quello di aggiungere un insieme di layer di Self-Attention in parallelo a quella semplice, denominato *head*. Per ogni calcolo vengono mantenute delle matrici di pesi separate per le query, le key e i value. Ogni head è inizializzato in maniera casuale ed è utilizzato per proiettate gli input in diverse rappresentazioni di sotto-sequenze spaziali.

Nel Transformer il numero di head è pari 8; ne conseguono 8 differenti matrici di output. Queste, dato che l'output deve essere una singola matrice, vengono concatenate assieme per poi essere moltiplicate con una matrice addestrata con l'intero modello. Il risultato dunque contiene informazioni su diverse componenti spaziali e aiuta il modello a concentrasi su differenti posizioni,come visibile in Figura 1.10.

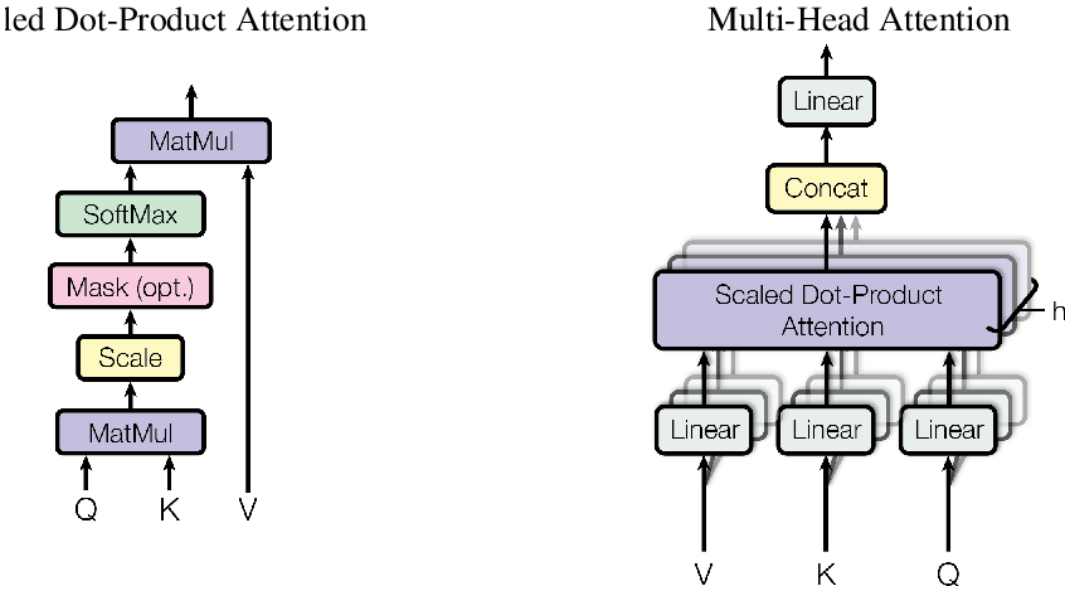


Figura 1.10: Multi-Head Attention [34]

1.3.4 L'architettura del Transformer

Come la maggior parte dei modelli di traduzione sequenziale, il Trasformer ha una struttura composta da un encoder e un decoder. Il primo di questi mappa una sequenza di input di simboli in una sequenza di rappresentazione continua z, mentre il secondo, dato z, ne ricava un output, un simbolo alla volta. L'architettura implementata da questo modello può essere visualizzata in Figura 1.11.

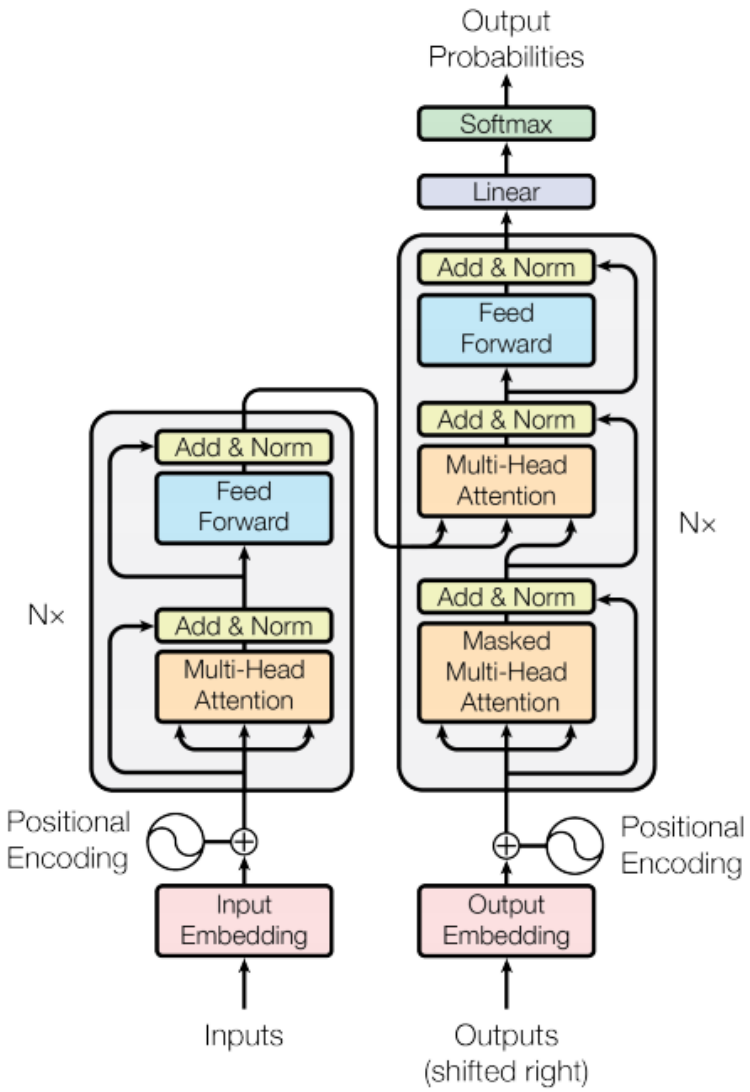


Figura 1.11: Architettura del Transformer [34]

Nella parte destra dello schema si può osservare la struttura base dell'encoder: esso è composto da 6 livelli di Multi-Head Self-Attention e da una rete feed-forward completamente conessa. L'output di ogni sottolivello è poi sommato e normalizzato seguendo la formula $LayerNorm(x + Sublayer(x))$ dove $Sublayer$ è la funzione implementata dal sottolivello. Ogni sottolivello produce un output di dimensione pari a 512.

Il decoder è composto da uno stack di 6 livelli identici. In aggiunta ai sottolivelli dell'encoder, si aggiunge un livello di multi-head Attention che

elabora l'output dell'encoder. Inoltre è stata modificato il livello di Self-Attention e spostato l'output dell'embedding di una posizione per far si che le previsioni di un elemento alla posizione i potessero dipendere dagli input output conosciuti in posizione minore di i.

1.4 BERT

Il Trasformer rappresenta una architettura innovativa alla base di molti modelli di rete neurali. Tra questi è presente anche BERT [9], acronimo di "Bidirectional Encoder Representation from Transformer". Pubblicato nel 2019, è progettato per offrire una architettura unificata per diversi task e per l'implementazione dell'approccio bidirezionale nel training. Di seguito si illustrano gli elementi distintivi di questo modello.

1.4.1 Il pre-addestramento di modelli di linguaggi e l'approccio bidirezionale

In BERT è presente una fase di preaddestramento su specifici task che porta alla formazione di un modello sul quale può essere fatto fine-tuning di diversi task, raggiungendo ottimi risultati. Questa idea non è nuova ma anzi, è stato dimostrato che il pre-addestramento di modelli di linguaggio permette di migliorare i risultati in molti task, in particolare quelli per l'elaborazione testuale.

In particolare esistono due approcci per l'implementazione di questo meccanismo.

- Feature-based. Vengono utilizzate architetture specifiche per un determinato task e si include il pre-addestramento come feature aggiuntiva. Questa tecnica è utilizzata in rete neurali come ELMO [20].

- Fine-tuning. Vengono introdotti pochi parametri specifici di un determinato task e l'addestramento viene eseguito su task comuni per poi effettuare il fine-tunining di tutti i parametri ottenuti. Questa tecnica è utilizzata dalla rete neurale OpenAI GPT [23].

Il secondo approccio viene generalmente implementato come modello unidirezionale: si considera solo il contesto precedente o solo il contesto successivo all'input considerato. Questo ha come conseguenza che si possono utilizzare un numero limitato di architetture per il pre-training e che si perdono informazioni sul contesto.

All'approccio unidirezionale si contrappone l'approccio bidirezionale. Per comprendere l'utilità e l'idea generale di quest'ultimo si consideri il seguente

esempio:
input:"The boy went to the baker and bought some bread"
task: prevedere la prova "baker"
Nell'approccio unidirezionale si considera solo il contesto precedente alla parola voluta: "The boy went to the".
Nell'approccio bidirezionale si considerano sia il contesto precedente che il successivo: "The boy went to the" e "and bought some bread".
È facilmente intuibile come nel secondo approccio sia più semplice prevedere la parola desiderata e come il significato della frase, nel primo caso, si possa comprendere solo al termine dell'analisi della frase. L'approccio bidirezionale è stato presentato per la prima volta nelle BLSTM [29]: esse sono basate su reti ricorrenti e quindi presentano i problemi illustrati precedentemente di perdita di informazione e rallentamento del calcolo. Il modello BERT utilizza l'approccio bidirezionale all'architettura del Transformer cercando di superare così i limiti delle reti ricorrenti ma mantenendo i vantaggi dell'approccio bidirezionale.

1.4.2 BERT e l'approccio bidirezionale

L'architettura di BERT è caratterizzata dal pre-addestramento del modello con il meccanismo del fine-tuning. Dunque sono presenti due step di addestramento: il pre-training e il fine-tuning. Nella prima fase il modello è addestrato su dati non etichettati e su differenti task, nella seconda si effettua il fine-tuning utilizzando dati etichettati e sui task più diffusi. L'approccio bidirezionale è implementato nella fase di pre-addrestamento con l'introduzione del *Masked Language Model*. Questo task molto semplice considera come input delle frasi all'interno delle quali sono state mascherate, appunto, delle parole e si addestra la rete a prevedere queste ultime. Con questo stratagemma la rete apprende informazioni sulla struttura delle frasi e riesce a considerare sia la parte precedente della frase che la parte successiva alla parola da prevedere.

Il meccanismo di mascheramento, in particolare, è il seguente:

- 80%: il token è sostituito con [MASK];
- 10%: il token è sostituito con uno casuale e questo permette l'apprendimento del significato semantico delle frasi;
- 15%: il token rimane invariato e permette alla rete di apprendere anche in frasi reali in cui non è presente il token speciale [MASK].

In questa fase la funzione di loss utilizzata è una cross-entropy sulla previsione dei token mascherati rispetto all'input effettivo.

L'architettura di BERT

Dagli elementi precedentemente analizzati è possibile comprendere la struttura di BERT come visibile in Figura 1.12.

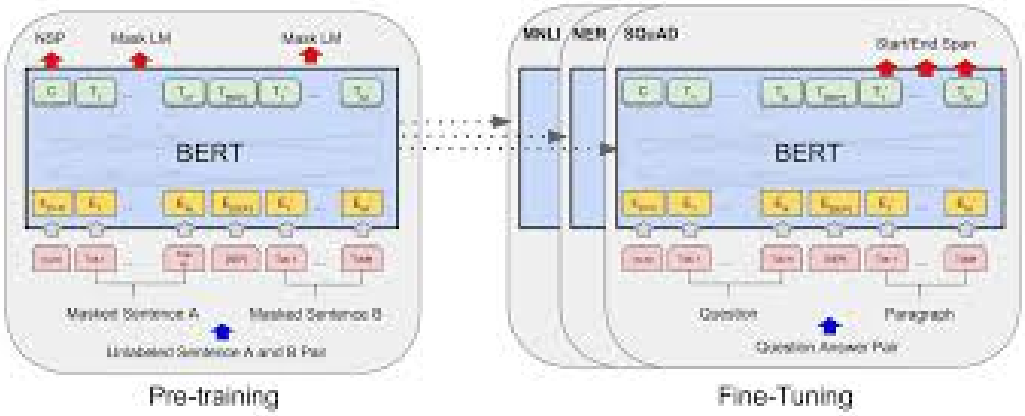


Figura 1.12: L'architettura di BERT [9]

Sia nella fase di pre-training che in quella di fine-tuning è utilizzata una architettura molto simile al Transfomer originale che comprende solo la parte di encoder e non il decoder. Nel pre-training vengono considerati i task di Masked Language Model, descritto precedentemente e il Next Sentence Prediction. Quest'ultimo permette di apprendere le relazioni fra le parole: si suddivide la frase di input in due sezioni, A e B, e si addestra la rete a predire B da A.
BERT utilizza il meccanismo dell'Attention per unire l'operazione di codifica delle coppie e l'applicazione dell'Attention su entrambi gli elementi: codificando le coppie di testo con la self-Attention di fatto calcola l'Attention bidirezionale fra le due frasi.

Successivamente, per ogni task, si immettono gli input specifici e si effettua il fine-tuning dei parametri end-to-end. I task considerati per il fine-tuning sono: accoppiamento di frasi, implicazioni fra ipotesi e conclusioni, question answering e text classification.

1.4.3 L'input del modello

BERT ha la caratteristica di presentare un'architettura unificata per diversi task. Questo porta a dover avere un input che sia versatile e che permetta di lavorare in diversi modi.

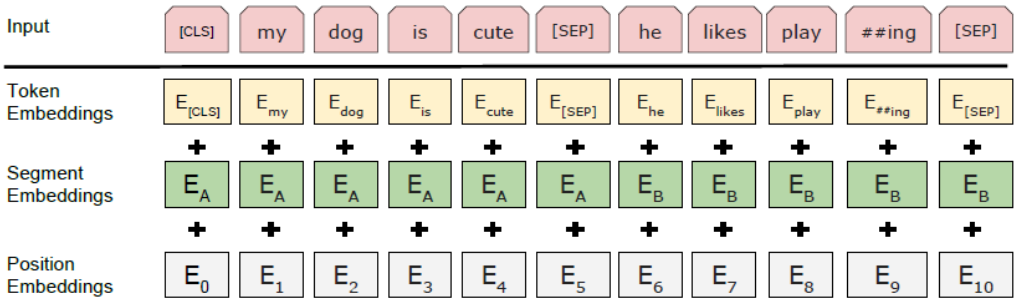


Figura 1.13: L'architettura di BERT [9]

Come visibile in Figura 1.13 l'input è costituito da tre componenti principali:

- il token embedding: trasformazione delle parole in token con il metodo scelto;
- il positional embedding: come detto nella sezione precedente, racchiude le informazioni di posizione assoluta e relativa dei token nella frase;
- il segment embedding: embedding che corrisponde alle due sezioni della frase. Questo permette di distinguere le sezioni in A e B per il task di Next Sentence Prediction.

Con queste componenti la rete riesce a lavorare sui diversi task della fase di pre-addestramento.

I modelli disponibili

Esistono diversi modelli e pesi pre-addestrati di BERT ma nel paper vengono presentate in particolare due strutture che si distinguono per numero di layer e di head della multi Attention. Le loro caratteristiche sono riassunte nella seguente Tabella 1.4.3.

	BERT_{BASE}	BERT_{LARGE}
Numero di layer nel trasformer L	12	24
Numero di Self-Attention head A	12	16
Dimensione dello stato interno h	768	1024
Numero totale di parametri	110M	340M
Lunghezza della sequenza	64	64
Dimensione massima dei batch	64	12
Addestrato con Cloud TPU	64GB di RAM	64GB di RAM

Inoltre esistono altre versioni che combinano questi parametri, per l'elenco coompleto si rimanda al link con il codice GitHub [25].

1.4.4 I risultati e limitazione di BERT

Bert, al momento della pubblicazione, ha migliorato lo stato dell'arte in diversi task di elaborazione del testo. In particolare nel paper [9] viene riportato che il modello ha raggiunto risultati minori su GLU rispetto ad altre reti neurali come ELMO, OpenAI GPT e i corrispondenti modelli di riferimento negli altri task.

Dunque questo modello costituisce un grosso passo avanti ma presenta anche alcune limitazioni:

- supporta input di lunghezza massima pari a 512 token;
- nel lavoro pubblicato, il modello è stato addestrato con 64 GB di RAM, utilizzando meno risorse non si ottengono gli stessi risultati;
- se si utilizzano 12-16 GB, come nelle normali GPU si presentano dei problemi con versione Large per la dimensione massima dei batch, 12, che richiede molta memoria. Quest'ultima richiede un quantitativo di risorse difficile da avere normalmente.

1.5 T5

Un altro modello basato sull'architettura del Transformer e che è stato preso in considerazione nel lavoro svolto è T5 [24]. Questo modello, acronimo di "Text-to-Text Transfer Transformer", si pone l'obbiettivo di poter lavorare su diverse tipologie di task raggiungendo notevoli risultati.

1.5.1 Un modello generico

Il concetto alla base di questo modello è il transfer-learning, che è stato analizzato precedentemente. Nel lavoro pubblicato dal gruppo di ricerca di Google [24] si effettua una analisi dei principali modelli che utilizzano questa tecnica per poi presentare un modello innovativo che prenda il meglio di ogni approccio: il risultato è T5.
Per applicare al meglio questo approccio una componente importante risulta essere il dataset utilizzato per l'addestramento. Esso deve essere di alta qualità, diversificato ma anche di dimensione imponente. A questo scopo è stato realizzato per questo lavoro il dataset C4, *Colossal Clean Crawled Corpus*. Esso è stato generato basandosi sul dataset disponibile on-line denominato Common Crawl [1] che rappresenta una repository open si dati web scansionati che possono essere acceduti da chiunque. Avendo come punto di partenza questo

dataset, gli autori di T5 hanno rielaborato i dati da vari task per riportarli nella forma text-to-text, come mostrato in Figura 1.14.

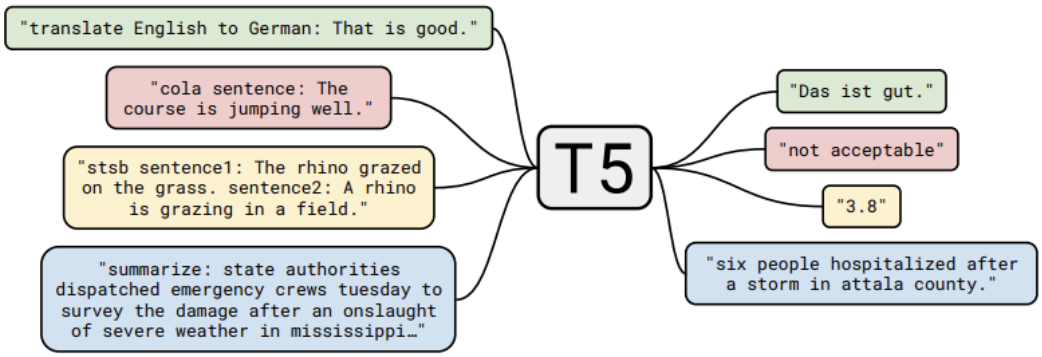


Figura 1.14: I task di T5 [24]

Il pre-training sul dataset C4 introduce alcune migliorie: permette alla rete neurale di apprendere un determinato grado di conoscenza sui linguaggi naturali e di addestrare il modello su task testuali arbitrari che presentano un testo sia in input che output come traduzione, riassunti, classificazione, regressione e comprensione testuale.

1.5.2 T5 e BERT

T5, come il suo predecessore BERT, è basato, come già illustrato, sul meccanismo del transfer-learning. Anche se l'intuizione alla base è la medesima, il team di Google ha introdotto con T5 alcune differenze. Di seguito si elencano le principali sia nell'architettura che nell'utilizzo:

- architettura:
  - BERT comprende solo la componente dell'encoder del tranformer;
  - T5 riproduce essenzialmente l'intera struttura encoder-decoder del Transform come visibile in Figura 1.15 [3].

[3]https://medium.com/analytics-vidhya/t5-a-detailed-explanation-a0ac9bc53e51

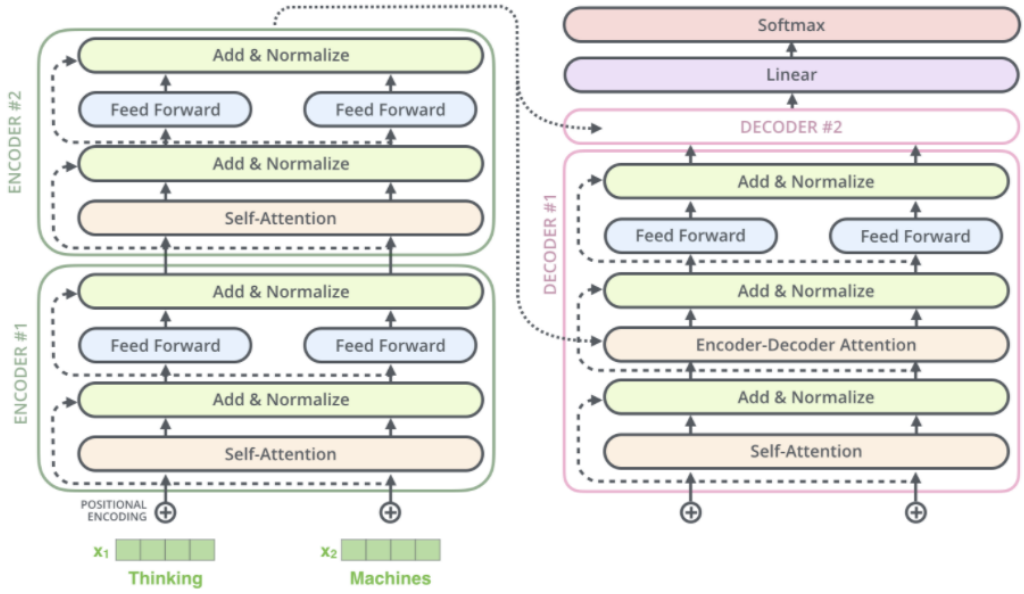


Figura 1.15: L'architettura di T5

- fine-tuning su specifici task:
  - BERT, per poter essere utilizzato per un determinato task, deve essere ampliato con layer aggiuntivo, dunque, per esempio, per una classificazione si aggiunge un layer che riporti il risultato della rete a quello della classificazione voluta;
  - in T5 la scelta del task da utilizzare è indicata semplicemente anteponendo una stringa con il nome del task all'input.

- addestramento di un nuovo task: con la caratteristica descritta precedentemente, in T5 può essere effettuato un addestramento su un nuovo task introducendo un nuovo prefisso all'input del dataset prescelto. In BERT questo comporta un cambio sostanziale nell'architettura;

- dimensione del modello: T5 è si più versatile ma ha una dimensione molto elevata e quasi triplicata rispetto a BERT.

In generale, con T5 si è realizzato un modello flessibile e estensibile con un qualsiasi task riportabile alla forma text-tot-text e, trattandosi di elaborazione testuale, questo può e dovrebbe essere molto semplice.

1.5.3 I modelli esistenti

Come nel caso di BERT, esistono modelli pre-addestrati diversi di T5 che si differenziano per il numero di livelli interni e per il numero di head. In generale il numero di pesi presente in questo modello è molto elevato e, di conseguenza, per l'addestramento o per l'utilizzo sono necessarie risorse computazionali consistenti. In Tabella 1.5.3 vengono riportati i modelli principali disponibili on HuggingFace [21].

modello	caratteristiche
t5-small	60M parameters with 6-layers, 512-hidden-state, 2048 feed-forward hidden-state, 8-heads, Trained on English text: the Colossal Clean Crawled Corpus (C4)
t5-base	220M parameters with 12-layers, 768-hidden-state, 3072 feed-forward hidden-state, 12-heads, Trained on English text: the Colossal Clean Crawled Corpus (C4)
t5-large	770M parameters with 24-layers, 1024-hidden-state, 4096 feed-forward hidden-state, 16-heads, Trained on English text: the Colossal Clean Crawled Corpus (C4)
t5-3B	2.8B parameters with 24-layers, 1024-hidden-state, 16384 feed-forward hidden-state, 32-heads, Trained on English text: the Colossal Clean Crawled Corpus (C4)
t5-11B	11B parameters with 24-layers, 1024-hidden-state, 65536 feed-forward hidden-state, 128-heads, Trained on English text: the Colossal Clean Crawled Corpus (C4)

1.5.4 I risultati e i limiti del modello

Questo modello, al momento della pubblicazione, ha raggiunto lo stato dell'arte in diversi dataset e problemi conosciuti GLUE, SQUAD, ENRO e altri [24].

La principale limitazione di questo modello è la quantità enorme di pesi e la sua grandezza stessa. Questo porta ad un numero di risorse necessario per l'addestramento che non è indifferente e potrebbe non essere disponibile normalmente. Gli stessi autori definiscono questo aspetto come limite e consigliano per l'addestramento l'utilizza delle risorse TPU di Google. Non rappresenta dunque un modello facilmente addestrabile.

Capitolo 2

I Transformer efficienti

2.1 I limiti del Transformer

Nel capitolo precedente si sono illustrate le innovazioni che hanno portato alla costruzione dell'architettura del Transformer. Nel corso del tempo sono stati introdotti diversi aspetti che sono poi stati inglobati in questa architettura:

- Bag-of-words: si convertono le informazioni testuali in forma numerica e quindi elaborabile;
- Word embedding: si estraggono il significato e delle intuizioni di esso dalle parole rappresentate;
- RNN: si estraggono le informazioni spaziali delle parole all'interno delle frasi;
- LSTM: si minimizza il problema della discesa del gradiente;
- LSTM bidirezionali: nella predizione di considera tutto il contesto della frase e non solo una parte di questa;
- Attention: si concentra la predizione su determinate posizioni assegnando ad esse dei pesi;
- Transformer: si parallelizza il calcolo per dati sequenziali, migliorandone le performance.

L'architettura del Transformer ha riscosso un incredibile successo grazie alla sua efficienza e applicabilità in una vasta gamma di domini. Essa non rappresenta il punto di arrivo della ricerca ma anzi, recentemente sono state introdotte diverse varianti dell'architettura base per migliorarne l'efficienza.

Per comprendere poiché il Transformer richiede delle ottimizzazioni occorre riprendere la formula dell'Attention:

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

Una delle componenti principali del calcolo è la funzione di softmax che è calcolata con la seguente formula:

$$Softmax(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)}$$

Per il calcolo del denominatore della formula è necessario avere a disposizione il risultato della moltiplicazione matriciale QK^T che rappresenta una matrice quadratica rispetto alla lunghezza dell'input. Questo porta l'intero calcolo a non scalare linearmente ma anzi quadraticamente rispetto all'input.

Di seguito si descrivono le ottimizzazioni introdotte per cercare di ovviare a questo problema per poi descrivere nel dettaglio il modello utilizzato in questo lavoro di tesi: il Performer.

2.2 I Transformer efficienti

Dal 2017, anno di presentazione del Transformer, sono state introdotte diverse architetture che hanno introdotto un grado di efficienza nel calcolo dell'Attention. Esso può essere individuato in tre aspetti differenti: impronta di memoria del modello, costo computazionale e efficienza nel processare sequenze di input lunghe. In tutti i casi l'obbiettivo è quello di introdurre una approssimazione nel calcolo della matrice dell'Attention applicando una determinata nozione di sparsità [30].

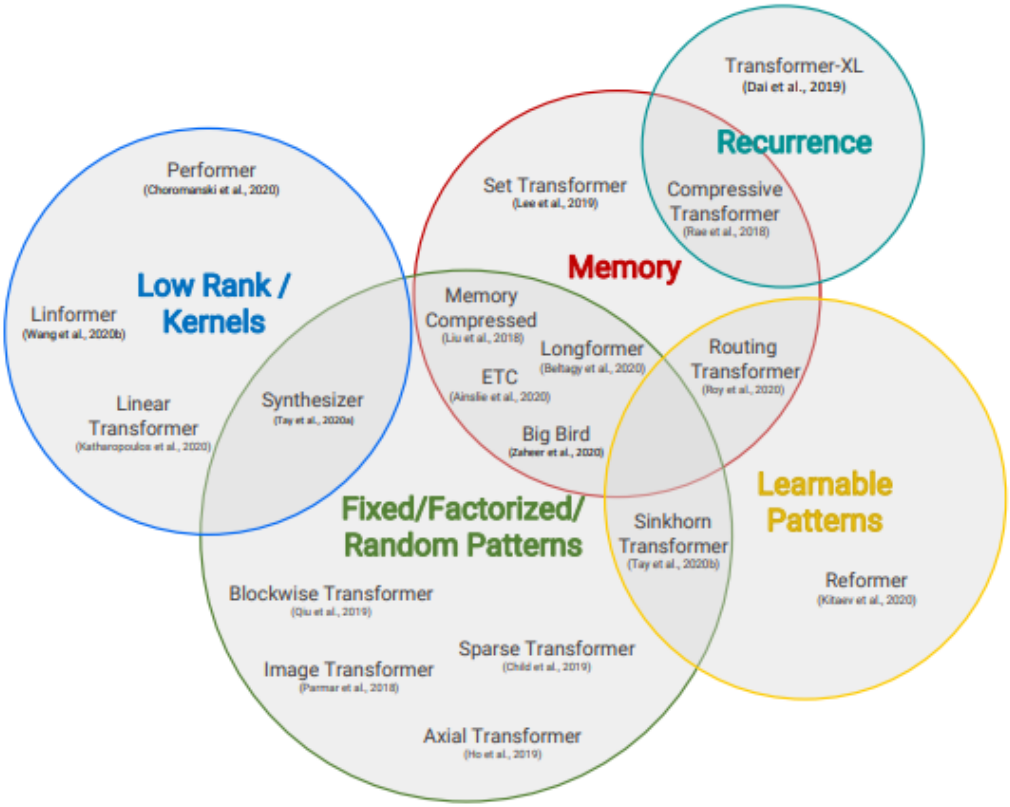


Figura 2.1: Tassonomia delle architetture efficienti di Transformer [30]

Come osservabile in Figura 2.1, esistono le diverse architetture si possono distinguere per la tecnica applicata nell'approssimazione. Di seguito si riporta una breve descrizione per ogni classe:

- fixed pattern: Limita il campo di vista a pattern fissi e predefiniti come le finestre locali. Tra i possibili pattern ci sono: Blockwise pattern, Strided pattern, Compressed pattern, o una combinazione di questi.

- pattern addestrati: si addestra con la rete il pattern da visualizzare. La funzione di loss utilizzata è molto simile a quella utilizzata dal clustering k-means e permette di dividere i pattern in gruppi.

- memoria: una sezione della memoria viene utilizzata come pattern globale e che immagazzina un contesto temporaneo ma condiviso da tutto il modello. Questo approccio è anche chiamato Attention parametrizzata.

- metodi low-rank: si approssima la matrice dell'Attention con il metodo del low-rank e quindi si migliora la complessità dovendo effettuare meno calcoli;
- kernel: si rivede il calcolo dell'Attention attraverso l'utilizzo delle funzioni kernel e anche in questo caso si approssima la matrice dell'Attention;
- ricorrenza: si condividono le informazioni nei vari blocchi con la tecnica della ricorrenza.

È importante notare come queste classi non escludano a vicenda. Per esempio, il Performer, che verrà descritto successivamente, utilizza sia il low-rank che le funzioni kernel.

2.3 I modelli di Transformer efficienti

Di seguito si elencano alcuni modelli che utilizzano le ottimizzazioni descritte precedentemente e che hanno raggiunto un discreto successo.

2.3.1 LongFormer

Il *Longformer* [4] rimpiazza il calcolo dell'Attention utilizzando due diverse tecniche di Attention: la window Attention e la selected global Attention.

- Windows Attention. Ogni token guarda se stesso e i suo vicini come nelle CNN. L'idea è quella che le parole strettamente vicine alla parola devono essere considerate come essenziali per il calcolo dell'Attention. Più ci si addentra nei livelli della rete, più il singolo token raccoglie informazioni su tutti i token. La complessità è pari a $O(n * k)$ con k pari alla dimensione della finestra e n la lunghezza dell'input;
- Selected Global Attention. Si definiscono dei token specifici che vengono considerati sempre poiché necessari. La sua complessità è pari a $O(n * m * 2)$, dove m sono i token speciali che devono essere considerati da e a ogni token di input, quindi moltiplicati per due.

La memoria totale richiesta è pari a $O(n * k + n * m * 2)$, o $O(n(k + m))$.

2.3.2 Reformer

Il *Reformer* [15] si basa sula LSH Attention, locality sensitive Attention, che introduce dei parametri condivisi fra le query e le key. Si applica una funzione di hash sia alle query che alle key e ci si basa sul principio che i vettori più

vicini avranno hash simili rispetto ai vettori che sono distanti.
Con questa operazione si selezionano i vettori da considerare per il calcolo dell'Attention: dato un input e il suo hash, si calcola l'Attention solo con gli altri input che hanno un hash uguale a esso. Di fatto si dividono gli input in gruppi basandosi sul loro valore di hash e questi raggruppamenti poi vengono utilizzati per il calcolo dell'Attention. Con questa tecnica la complessità si riduce a $O(nlog(n))$.

2.3.3 Linformer

Il *Linformer* [35] approssima la matrice di Attention in una matrice low-rank. Nel lavoro presentato, si dimostra infatti che oltre il 90% delle variazioni nei pesi della matrice di Attention può essere compreso dai primi 128 di 512 autovalori della matrice. Quindi ma maggior parte dell'informazione è concentrata in poche dimensioni che hanno gli autovalori maggiori. Con questo modello è complesso mantenere la causalità e prevenire la fusione fra input passati e futuri. Dall'altra parte la complessità è pari a $O(n * k)$ dove k è la lunghezza delle proiezioni e di solito, nella pratica, è un valore molto basso che non inficia sulle performance considerabili quindi come $O(n)$.

2.3.4 Big Bird

Uno dei più moderni sistemi di analisi testuale basato su machine learning è *Big Bird*, [38]. Questo modello è stato pubblicato a Luglio 2020 da Google e, in questo momento, il codice realizzato non è ancora stato messo a disposizione della comunità. Big Bird introduce molte innovazioni e migliora drasticamente le performance in vari task NPL come la question answering e la scrittura di riassunti.
Il punto di partenza per lo sviluppo di questo modello, è rappresentato dal concetto già presentato da modelli precedenti, come BERT (citazione papar BERT), di Self-Attention e il miglioramento delle sue performance e della quantità di operazioni che è necessario svolgere, migliorando la complessità attuale di $O(n^2)$. Dunque è stato realizzato il meccanismo di $SparseAttention$ che raggiunge gli stessi risultati con un numero di prodotti scalari pari a $O(n)$. Così facendo sono stati raggiunti i seguenti risultati:

- il modello soddisfa tutte le proprietà del Transformer con Self-Attention;
- è Turing equivalente;
- raggiunge o migliora l'attuale stato dell'arte nei principali task di text-analysis;

- grazie al ridotto numero di operazioni e alle minore risorse computazionali richieste è in grado di considerare un input più grande di 8 volte e quindi considerare più contesto.

Di seguito si riportano le principali caratteristiche di questo modello.

L'ottimizzazione del meccanismo dell'Attention

La struttura di Big Bird è basta su alcuni fondamenti teorici derivanti dalla teoria dei grafi che permettono di effettuare un numero ridotto di operazioni e quindi migliorare le performance.
Per comprendere la struttura di base si deve analizzare il meccanismo di generalised Attention. Questo meccanismo può essere descritto da un grafo D i cui vertici rappresentano i gli input mentre gli archi i prodotti scalari da effettuare per calcolare la funzione di Attention. Il grafo può essere rappresentato con la propria matrice di adiacenza associata e il numero di celle con valore 1 rappresenta il numero di prodotti da calcolare. In base al contenuto delle celle della matrice, si ottiene un meccanismo differente: se, per esempio, ogni cella ha al suo interno un 1, allora si ottiene la Self-Attention di BERT, come è possibile osservare in Figura 2.2.

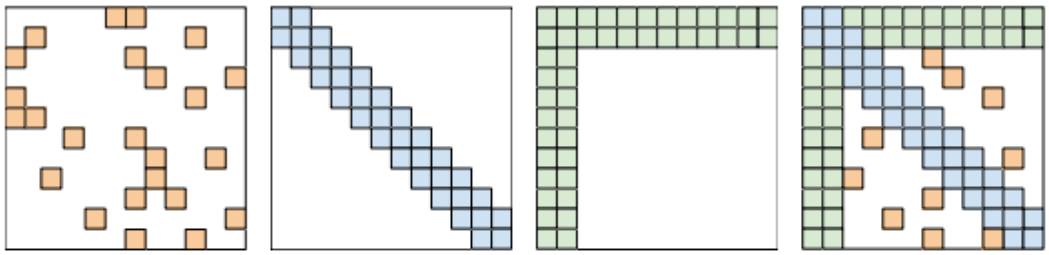


Figura 2.2: Meccanismi di Attention nella matrice di adiacenza [38]

Con questa rappresentazione, il problema di riduzione della complessità diventa un problema di sparsificazione sul grafo e l'obbiettivo è trovare un grafo sparso che permetta di raggiungere le stesse performance con un numero minore di operazioni dunque avere: una media delle distanze minime fra i nodi del grafo bassa e una nozione di località.

I grafi random

La prima condizione viene soddisfatta considerando i grafi sparsi del modello di Erdös-Rényi. Questi grafi sono costruiti scegliendo i maniera casuale un numero di archi pari al numero di nodi presenti nel grafo. E' noto che, con

una struttura di questo tipo, il cammino medio minimo che separa due nodi del grafo è pari al logaritmo del numero dei nodi stessi. Come conseguenza questo grafo approssima il grafo completo mantenendo la quantità principale di informazione.

Il principio di località

Si è notato che nelle principali tipologie di task NPL di analisi del testo i dati hanno la caratteristica di avere un grado di località molto elevato e quindi la maggior parte dell'informazione su un token può essere derivata dai suoi vicini. Come misura del grado di località, è stato considerato il coefficiente di clustering che, nella teoria dei grafi, rappresenta il numero di sotto-grafi maggiormente interconnessi: più esso è elevato maggiore è il valore del coefficiente. I grafi di Erdös-Rényi non hanno questa caratteristica e quindi sono stati scartati a favore del modello di Watts e Strogatz che presenta una bassa distanza minima media fra i nodi e un buon grado di località. Questi grafi sono costruiti partendo da una struttura regolare ad anello con n nodi connessi con w vicini, $w/2$ a destra e $w/2$ a sinistra. Di questi archi un certo sottoinsieme ($k\%$) è rimpiazzato con collegamenti random.
In Big Bird, questa struttura è implementata con un meccanismo di sliding window Attention: durante la Self-Attention di larghezza w, la query alla posizione i considera da $i - w/2$ a $i + w/2$ chiavi. E quindi nella matrice di adiacenza le celle con indice $(i, i - w/2 : i + w/2)$ hanno valore pari a 1.

I global token

L'ultimo elemento della struttura di Big Bird sono i global token. Durante la prima fase di sperimentazione, i ricercatori di Google, hanno trovato che, applicando solo la struttura del grafo sparso e regolare, non si riuscivano ad eguagliare i risultati ottenuti con BERT. Dunque si è deciso di utilizzare un maggior numero di archi introducendo i global token: dei token che vengo sempre considerati per il calcolo dell'Attention. Essi sono individuati su due livelli:

- BigBird-ITC: nella costruzione interna del Transformer è individuato un sottoinsieme di indici G tali per cui, nella matrice di adiacenza, $(i, :)$ e $(:, i) = 1$ per ogni $i \in G$;
- BigBird-ETC: si includono token aggiuntivi come CLS. Se crea una nuova matrice di adiacenza, aumentando le dimensioni della precedente e aggiungendo un insieme di token G tale che $(i, :)$ e $(:, i) = 1$ per ogni $i \in G$. Essi rappresentano posizioni aggiuntive per contenere informazioni sul contesto dell'input.

Con questa costruzione finale si ottengono le 3 proprietà del modello Big Bird:

- le query vengono confrontate con un numero r di chiavi casuali ;
- ogni query comprende $w/2$ token a destra e $w/2$ token a sinistra per rendere il modello bidirezionale;
- gli w token contengono g global token (token esistenti o aggiuntivi).

Questo modello è stato testato sui principali problemi di text mining come: question answering, document classification e summarization.

2.4 Il Performer

In questa sezione si si analizza più nello specifico la variante dell'Attention quadratica che è stata utilizzata per questo progetto di tesi: il **Performer** [8].

2.4.1 Il kernel trick

Un elemento teorico che è alla base del Performer è il *kernel trick*. Il Kernel [3] è un metodo di calcolo del prodotto scalare di due vettori in uno spazio di feature, possibilmente di dimensionalità elevata, e per questo viene anche chiamato "generalized dot product".

Si supponga di avere un mapping $\phi : \mathbb{R}^n \to \mathbb{R}^m$ che traspone i vettori da uno spazio $\mathbb{R}^n$ a uno spazio $\mathbb{R}^m$ e il prodotto di due vettori x e y in questo spazio pari a $\phi(x)^T\phi(y)$. Un kernel è una funzione che corrisponde a questo prodotto scalare: $k(x, y) = \phi(x)^T\phi(y)$.

Una funzione kernel è utile poiché riesce a calcolare questo prodotto scalare senza trasporre i vettori nel nuovo spazio e senza sapere cosa sia ϕ.

Per comprendere meglio questo concetto si consideri il kernel polinomiale $k(x, y) = (1 + x^T y)^2)$ con $x, y \in \mathbb{R}^2$. Si assuma che $x = (x_1, x_2)$ e $y = (y_1, y_2)$ e si espanda l'espressione precedente:

$$k(x, y) = (1 + x^T y)^2) = (1 + x_1y_1 + x_2y_2)^2 =$$
$$1 + x_1^2y_1^2 + x_2^2y_2^2 + 2x_1y_1 + 2x_2y_2 + 2x_1y_1x_2y_2$$

Questo non rappresenta altro che un prodotto scalare fra due vettori e può permettere, per esempio, il calcolo di esso in uno spazio a sei dimensioni senza esplicitare questo spazio. Il kerner trick corrisponde a questa operazione: calcolare una funzione corrispondente al prodotto in un spazio vettoriale di dimensione maggiore, senza effettivamente eseguire la mappatura dei vettori in esso.

Un altro esempio di kernel è quello Gaussiano $k(x, y) = exp(-\gamma||x - y||^2)$. Questa funzione ha la caratteristica fondamentale che, se espansa come sviluppo di Taylor, corrisponde ad un codominio di dimensione infinita per γ.

Dunque, come possono essere applicate queste funzioni alle reti neurali [39]? Il primo utilizzo del kernel nel machine learning è costituito dalle Support Vector Machine, SVN, nelle quali il kernel trick è ampiamente utilizzato per far diventare problemi non-lineari, lineari. La loro applicazione principale è nella classificazione, nella quale la rete apprende come dividere gli input in gruppi definendo dei bordi di decisione. In questo task, come riportato in Figura 2.3, aumentando la dimensionalità dello spazio, è possibile dividere gli input con iper-piani lineari, operazione non possibile se si rimanesse nello spazio di input.

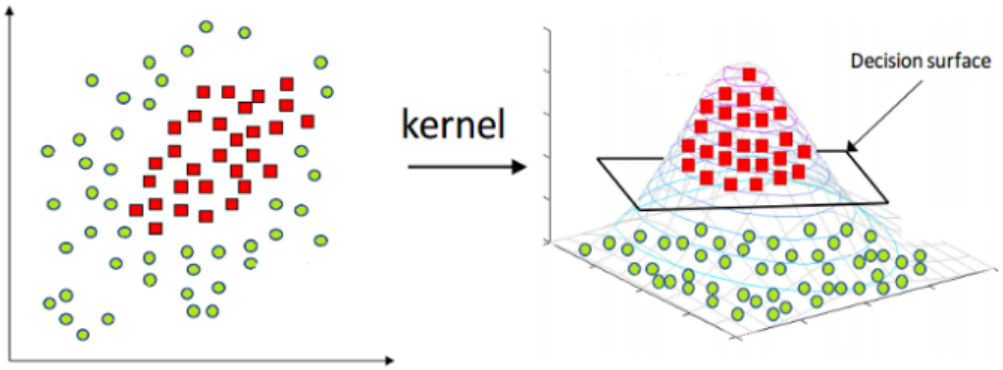


Figura 2.3: Esempio di aumento della dimensionalità dello spazio [39]

Il calcolo effettivo dei vettori in questo piano di dimensionalità maggiore può essere molto costoso dal punto di vista computazionale; per questo motivo le funzioni di kernel sono fondamentali perché permettono di calcolare una funzione senza eseguire effettivamente la trasposizione con una complessità complessiva lineare.

2.4.2 La risoluzione del collo di bottiglia della softmax

Nel contesto delle funzioni di kernel si inseriscono i Performer.

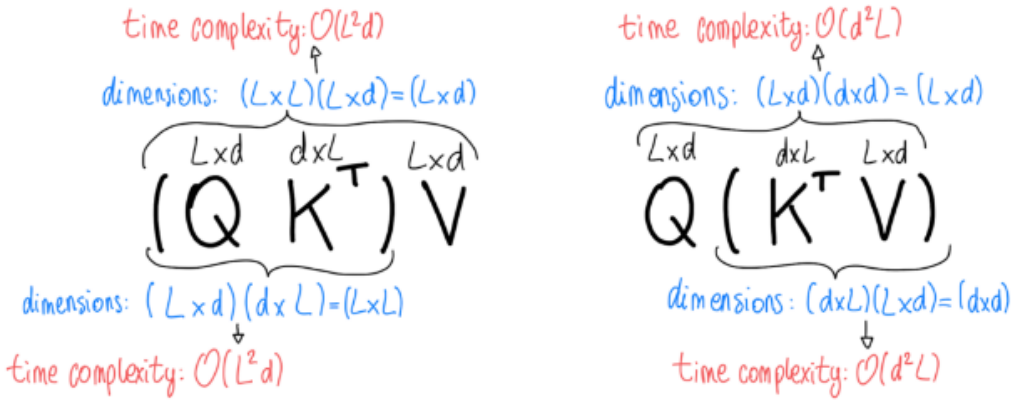


Figura 2.4: Complessità del calcolo dell'Attention [39]

In Figura 2.4 è osservabile la formula semplificata del calcolo dell'Attention che non considera la funzione di softmax. Da questo punto di vista è possibile notare come invertendo l'ordine di esecuzione delle moltiplicazioni tra matrici per ottenere una complessità minore. In particolare moltiplicando QK^T si ottiene una complessità di $O(L^2d)$, mentre se si effettua prima il prodotto K^TV si ottiene invece $O(d^2L)$. Nel secondo caso il valore ottenuto è di molto minore se si considera d come un iperparametro che è scelto di solito con un valore minore di L.

Nella formula classica, però, questa inversione dell'ordine non è possibile poiché è presente la funzione di softmax che costringe a gestire la complessità quadratica del calcolo della matrice QK^T.

La funzione di softmax dunque rappresenta il collo di bottiglia dell'Attention. Esiste un modo di approssimare il suo calcolo che permetta di scegliere l'ordine delle moltiplicazioni tra matrici? È possibile individuare due nuove matrici Q' e K' tali che: $Q'K'^T \approx softmax(\frac{QK^T}{\sqrt{d_k}})$?

2.4.3 FAVOR+

Al fine di individuare come effettuare l'approssimazione $Q'K'^T \approx softmax(\frac{QK^T}{\sqrt{d_k}})$ si utilizzano appunto, le funzioni kernel. In particolare i meccanismo introdotto nel Performer prende il nome di FAVOR+ [8] e di seguito di andranno ad illustrare le sue componenti principali.

In FAVOR+, la funzione di softmax è considerata come la funzione di kernel non calcolabile e l'obbiettivo è individuare la funzione ϕ che permetta di trasformare i vettori iniziali Q e K in Q' e K'. Quindi con la nozione

precedente: $k(x, y)$ corrisponde alla softmax, $\phi(x)^T$ è pari a Q e $\phi(y)$ è pari a K. È necessario, dunque, individuare ϕ.

In generale la funzione ϕ di un Kernel ha la seguente forma:

$$\phi = \frac{h(x)}{\sqrt{m}}(f_1(\omega_1^T x), ..., f_1(\omega_m^T x), ..., f_l(\omega_1^T x), ..., f_l(\omega_m^T x))$$

Dunque il vettore x si moltiplica per una distribuzione di vettori $\omega_1, ..., \omega_m$ e il risultato di ogni singolo prodotto viene dato in input alle funzioni scelte $f_1, ..., f_l$. Infine i vettori vengono moltiplicati per il fattore regolatore $\frac{h(x)}{\sqrt{m}}$.

Definendo $h(x), \omega_i$ e f_i si identifica la funzione kernel con la quale si approssima ϕ.

Il risultato teorico maggiore del lavoro che si accompagna al Performer è l'individuazione di questi elementi per approssimare la softmax (**FAV**):

$$h(x) = \frac{1}{\sqrt{2}} exp(\frac{-||x||^2}{2}),\ l = 2, f_1(u) = exp(u), f_2(u) = exp(-u)$$
$$D = Unif(\sqrt{d}S^{d-1})$$

L'utilizzo di una funzione kernel per il calcolo della softmax non è una novità assoluta, ma quella proposta con i Performer introduce alcuni elementi di novità.

- Con la distribuzione di ω scelta e assicurandosi l'ortogonalità dei vettori si riduce l'errore di approssimazione (**O**). I vettori ortogonali rappresentano una base per uno spazio e dunque riducono il rango della matrice da considerare diminuendo il numero di informazioni da che effettivamente vanno considerate. Inoltre il meccanismo di ORF permette di poter scegliere valori del rango della matrice approssimata più piccoli e dunque avere vettori di proiezione minori e una grandezza del modello in generale più piccola.

- Essa produce solo valori positivi (**R**+) che riducono la varianza e l'errore di approssimazione dato che i valori di Q e K originali sono sempre positivi.

Un altro apporto del lavoro è stata l'individuazione di un funzione stabile di kernel anche per la funzione di ReLU. Questo rende le performance ancora più elevate sia in accuratezza che nell'abbassamento dell'uso della memoria e del tempo di calcolo.

2.4.4 Il Performer

Il modello che utilizza il meccanismo FAVOR+ è appunto il Performer il cui codice è stato pubblicato [1]. Esso è applicabile alla maggior parte dei modelli esistenti e ha come parametro principale il numero di feature random da considerare. Esso, secondo quando indicato nel paper [8], deve essere pari a $d * log(d)$, con d pari alla dimensione di ogni head dell'Attention e corrisponde al rango della matrice approssimata dell'Attention. Inoltre in questo modello, il meccanismo del calcolo approssimato dell'Attention, viene utilizzato con l'algoritmo riassunto formalmente in Figura 2.5 nel quale la funzione di loss utilizzata è la cross-entropy e si calcola la differenza fra la frase di input e la frase di output.

Algorithm 1: FAVOR+ (bidirectional or unidirectional).

Input : $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{L \times d}$, isBidirectional - binary flag.
Result: $\widehat{\mathrm{Att}}_{\leftrightarrow}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \in \mathbb{R}^{L \times L}$ if isBidirectional, $\widehat{\mathrm{Att}}_{\rightarrow}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \in \mathbb{R}^{L \times L}$ otherwise.
Compute $\mathbf{Q}'$ and $\mathbf{K}'$ as described in Section 2.2 and Section 2.3 and take $\mathbf{C} := [\mathbf{V} \quad \mathbf{1}_L]$;
if isBidirectional **then**
  $\mathrm{Buf}_1 := (\mathbf{K}')^\top \mathbf{C} \in \mathbb{R}^{M \times (d+1)}, \quad \mathrm{Buf}_2 := \mathbf{Q}' \mathrm{Buf}_1 \in \mathbb{R}^{L \times (d+1)}$;
else
  Compute $\mathbf{G}$ and its prefix-sum tensor $\mathbf{G}^{\mathrm{PS}}$ according to (11);
  $\mathrm{Buf}_2 := \left[\mathbf{G}^{\mathrm{PS}}_{1,:,:} \mathbf{Q}'_1 \quad \ldots \quad \mathbf{G}^{\mathrm{PS}}_{L,:,:} \mathbf{Q}'_L\right]^\top \in \mathbb{R}^{L \times (d+1)}$;
end
$[\mathrm{Buf}_3 \quad \mathrm{buf}_4] := \mathrm{Buf}_2, \quad \mathrm{Buf}_3 \in \mathbb{R}^{L \times d}, \quad \mathrm{buf}_4 \in \mathbb{R}^L$;
return $\mathrm{diag}(\mathrm{buf}_4)^{-1} \mathrm{Buf}_3$;

Figura 2.5: L'algoritmo del Performer [8]

Infine è importante sottolineare che nell'algoritmo è previsto anche il calcolo della versione causale dell'Attention. Essa può essere utilizzata in problemi di natura generativa e supera altri modelli, come il Linformer, che non prevedono questa possibilità. Al fine del calcolo dell'Attention unidirezionale e causale si procede per step successivi eseguendo prima la moltiplicazione passo per passo degli elementi di K e di V, per poi passare alla moltiplicazione per Q.

[1] https://github.com/google-research/google-research/tree/master/performer

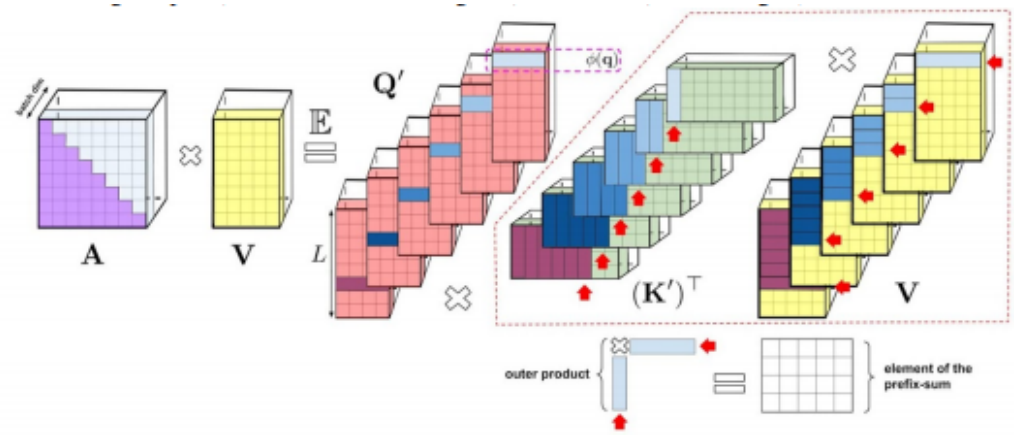


Figura 2.6: L'algoritmo del Performer per Attention causale [8]

L'algoritmo, come visibile in Figura 2.6, mantiene una somma che è una matrice ottenuta sommando i prodotti fra le feature random corrispondenti alle chiavi con vettori. Ad ogni iterazione dell'algoritmo, una feature random è moltiplicata per la più recente somma, ottenuta sommando tutti i prodotti relativi ai precedenti token, per ottenere una nuova matrice che rappresenta l'ouput dell'Attention.

2.4.5 I vantaggi del Performer

Per riassumere, nel lavoro presentato dal team di Google sono stati dimostrati i seguenti risultati che illustrano i vantaggi di questo modelli.

- Complessità in spazio e tempo: si dimostra che l'utilizzo del metodo FAVOR+ si avvicina al valore ottimo, ovvero il risultato della Attention quadratica, riducendo la complessità in spazio di $O(Lr + Ld + rd)$ e nel tempo di $O(Lrd)$. In questo modo il tempo di addestramento e di applicazione del modello risultano ridotti.

- Approssimazione del risultato: non viene applicata ma si sfrutta il meccanismo delle funzioni kernel.

- Dimostrazione teorica della sua potenza: viene illustrata una spiegazione matematica con relativa dimostrazione della potenza e dell'espressività del modello. Questo permette di spiegare in maniera chiara e con una dimostrazione il perché di alcuni risultati e di alcuni previsioni ottenute, operazione fondamentale se si applica il modello al contesto medico, dove è importante, più che altrove, comprendere le motivazioni dietro a determinate scoperte e azioni da svolgere.

- Compatibilità con i modelli attuali: il Performer può essere applicato ai principali modelli di Transformer presenti attualmente.
- Disponibilità del codice: il codice del kernel è stato rilasciato ed è accessibile su GitHub;
- il tempo di addestramento è minore e la lunghezza dell'input considerabile è maggiore.

Tutte queste caratteristiche hanno portato alla scelta di applicare questa ottimizzazione a BERT e T5 al fine di riuscire ad addestrare modelli di elevate dimensioni con risorse limitate.

2.5 Un confronto generale

In generale [30] ogni modello proposto in letteratura presenta pro e contro e, soprattutto, è incentrato su un aspetto o una tipologia di task specifica. In particolare, alcuni modelli si incentrano sulla modellazione di sequenze auto-regressive task generativi (Sparse Transformer, Adaptive Transformer, Routing Transformer, Reformers) altri su task come il question answering e la comprensione di testi (ETC e Linformer che non possono essere utilizzati nel caso precedente) e alcuni costituiscono un bilanciamento dei due (LogFormer). Infine dei modelli sono stati testati su domini specifici come la sintesi di proteine, Performer, e traduzione, Linear Transformer. È dunque comprensibile come un confronto generale sia difficile da effettuare.

Per concludere questa panoramica, si citano altre ottimizzazioni che sono state introdotte e che non comprendono l'ottimizzazione del calcolo dell'Attention.

- Condivisione dei pesi utilizzata nei modelli Universal Transformer e Quaternion Transformer;
- precisione mista: tecnica per ridurre il costo della memoria. Knowledge distillation: comprende il trasferimento della conoscenza di un molto complesso ad un modello più piccolo e facilmente addestrabile.
- Neural Architecture Search: ricerca generale di una struttura più performante dei Transformer e che possa superarne le performance.
- Addestramento per diversi task: si addestra un modello su un singolo task in maniera tale da poter riusare il modello su task differenti senza ulteriore utilizzo di risorse.

Le ottimizzazioni analizzate hanno lo scopo di rendere questi modelli estremamente complessi e difficili da addestrare accessibili ad una platea di ricercatori e studiosi più ampia e quindi far progredire la ricerca non solo nei laboratori delle grandi aziende, come Google, ma anche nei centri di ricerca universitari di tutto il mondo.

Capitolo 3

I modelli e le tecnologie

In questo capitolo si descriveranno nel dettaglio il framework e le architetture di implementazione dei modelli BERT e T5, i loro vantaggi e i motivi che hanno portato al loro utilizzo. Inoltre si definiranno gli elementi principali dell'implementazione fatta, ponendo le basi per poi descrivere al meglio i task e il caso applicativo preso in considerazione.

3.1 La libreria FLAX

L'implementazione dei modelli considerati, BERT e T5, sono stati implementati con FLAX [26]. Di seguito si illustrano le caratteristiche principali e i vantaggi che hanno portato all'adozione di questa libreria.

3.1.1 Da Autograd e XLA a JAX

Flax è un libreria specifica per l'implementazione di reti neurali ad alte prestazioni basata su JAX. Per comprenderne le caratteristiche è fondamentale spiegare le caratteristiche di JAX [1] e del suo antenato Autograd [2].

Autograd

Autograd è una libreria per il calcolo differenziale automatico. In particolare:

- è in grado di eseguire una differenziazione automatica di codice nativo Python e di Numpy;
- è progettato per eseguire in maniera efficiente derivate anche di grado maggiore a uno;

[1] https://github.com/hips/autograd
[2] https://github.com/HIPS/autograd

- supporta la differenziazione in modalità inversa, backpropagation, e quindi può elaborare in maniera efficiente i gradienti di funzioni con valori scalari rispetto agli argomenti con valori di matrice, oltre alla differenziazione in modalità diretta;

- la differenziazione diretta e quella inversa possono essere composte arbitrariamente;

- effettua una ottimizzazione sulle operazioni basate sul gradiente.

Questa libreria costituisce una innovazione importante ma attualmente i suo sviluppatori si sono spostati su JAX, di cui Autograd rappresenta la base.

XLA

Un altro componente di JAX è costituito da XLA [3], "Accelerated Linear Algebra". Esso è un compilatore specifico di dominio per l'algebra lineare che può accelerare i modelli di TensorFlow fino a 1.5 volte e, potenzialmente, senza apportare nessuna modifica al codice sorgente.

Quando un programma scritto in Tensorflow viene eseguito, le operazioni sono fatte individualmente da un executor di TensorFlow. Ogni operazione di TensorFlow ha una implementazione kerner per GPU precompilata che l'executor utilizza. Con XLA si introduce un'alternativa per eseguire i modelli: il grafo di esecuzione di TensorFlow viene compilato in sequenze di kernel di computazione generati specificatamente per il modello dato. Dato che questi modelli sono unici, possono applicare ottimizzazioni specifiche in base alle informazioni che ricevono. Di seguito si riporta un esempio per comprendere meglio questo meccanismo. Si consideri il seguente codice:

```
def model_fn(x, y, z):
   return tf.reduce_sum(x + y * z)
```

Senza l'utilizzo di XLA, il grafo di esecuzione lancia tre kernel: uno per la moltiplicazione, uno per l'addizione e uno per la reduce. Con XLA invece, viene computato il risultato in un unico kernel: le tre operazioni vengono fuse in un unico kernel GPU. Questa operazione non scrive in memoria i risultati i risultati intermedi prodotti dalle operazioni di moltiplicazione e somma ma li trasmette direttamente all'utente mantenendoli interamente nei registri di GPU. Questa operazione di fusione aumenta le performance rimuovendo delle operazioni da fare in memoria.

[3] https://www.tensorflow.org/xla

3.1.2 JAX

Dalle librerie di Autograd e LAX è stato implementato JAX che rappresenta un sistema per trasformare le funzioni numeriche. Di seguito vengono riportate le principali.

- Grad. JAX importa l'API di Autograd e la sua più famosa funzione grad per il calcolo di gradienti sia in versione diretta che inversa. Grad ha la funzione di calcolare il gradiente e le derivate di primo e di grado superiore per ogni funzione.

- JIT. Con XLA è possibile compilare le funzioni end-to-end con il decoratore jit; questo metodo permette di migliorare le performance dei programmi interpretati poiché la compilazione viene fatta durante l'esecuzione del programma, run time. Inoltre il compilatore jit ha un accesso dinamico alle informazioni dinamiche di runtime e questo permette di introdurre delle ottimizzazioni specifiche e più efficaci.

- Vmap. Questa funzione migliora la mappatura di funzioni in vettori e matrici. Essa migliora le prestazioni inserisce il loop nella primitiva di esecuzione della funzione.

- Pmap. Al fine di scrivere programmi che possano essere eseguiti su multipli processori o su più GPU si utilizza la funzione pmap. Quando la si applica si definisce che il codice interessato sia compilato con XLA e la sua esecuzione replicata e eseguita in parallelo nei ari device.

3.1.3 FLAX

FLAX rappresenta un continuo delle librerie precedenti e ne incorpora le principali funzioni, migliorandole per alcuni aspetti. Inoltre introduce determinati aspetti per lo sviluppo veloce di reti neurali. Questa libreria ha le seguenti componenti:

- una API per lo sviluppo di reti neurali, flax.linen;

- una classe per gli ottimizzatori, falx.optim;

- delle classi di utilità per il training replicato, la serializzazione, il checkpointing e le metriche;

- esempi di implementazione per i principali modelli;

- esempi per modelli veloci, grandi e distribuiti.

Il package flax.linen

In questo package vengono definiti gli elementi principali per la costruzione di una rete neurale. In particolare, la classe Module è la principale: tutti i layer e i modelli devono ereditare da essa. Questa classe offre i principali metodi per un rete neurale come: __call__, per eseguire il forward pass, e __init__ per l'inizializzazione del modello.

Gli altri package

Oltre al package linen, in FALX ne sono presenti altri. In particolare il package optim contine le classi per gli ottimizzatori per il passo di forward e per la discesa del gradiente, in struct si trovano le funzioni per costruire classi specifiche per le trasformazioni in jax e, nel pacchetto utils, si trovano tutte le principali funzioni di metriche, checkpoint e altre utili per lo sviluppo.

3.1.4 I vantaggi di FLAX

Questa libreria è molto innovativa e introduce notevoli miglioramenti. Il suo utilizzo però non risulta semplice e ha una curva di apprendimento, soprattutto se non si conoscono le librerie su cui è basata, molto alta. Una volta compresi i meccanismi principali però permette un facile e veloce sviluppo di modelli complessi e flessibili. Un altro vantaggio essenziale è la possibilità di implementare reti neurali che possono essere addestrati su più dispositivi e con calcolo distribuito. Questa caratteristica risulta fondamentale date le sempre più grandi dimensioni e numero di parametri che costituiscono i modelli di reti neurali come BERT e T5. Il calcolo parallelo permette appunto di non avere le stesse risorse e super computer ma comunque riuscire a sperimentare su questi modelli. Infine, un altro aspetto che ha portato alla scelta di questo framework, è stato cercare di rimanere il più possibile fedeli all'implementazione del Performer pubblicata dal gruppo di ricerca di Google, che come prima implementazione ha utilizzato proprio FLAX.

Nelle sezioni 3.2, 3.3 e 3.4 si descriveranno brevemente le cartelle dei codici sorgenti utilizzate come punto di partenza per il lavoro di ricerca.

3.2 Il Performer

Con la pubblicazione del paper sul Performer, il gruppo di ricerca di Google ha rilasciato il codice corrispondente su GitHub [4]. L'implementazione di FAVOR+ è attualmente disponibile in due versioni: una prima pubblicata a

[4] https://github.com/google-research/google-research/tree/master/performer/fast_attention

Novembre 2020 in JAX e una seconda per TensorFlow pubblicata in a fine Dicembre 2020.

In questa cartella è possibile trovare l'implementazione vera e propria della Attention approssimata proposta. In particolare sono disponibili due funzioni fondamentali che permettono l'inizializzazione della tipologia di Attention voluta:

- `make_fast_softmax_attention`: mette a disposizione una versione approssimata e unbiased della Softmax Attention regolare. Essa può essere utilizzata nel Transformer o da sola per ottimizzare la funzione di Softmax;

- `make_fast_generalized_attention`: implementa la Generalized Attention che produce diversi kernel per l'approssimazione dell'Attention. Nel lavoro viene utilizzata per sfruttare il kernel della funzione di attivazione ReLU.

Le due funzioni sono fondamentali per rendere il meccanismo FAVOR+ compatibile con le implementazioni di Transformer che utilizzano la Softmax Attention. Esse infatti hanno come output una funzione attention_fn che ha la medesima API di `flax.nn.attention.dot_product_attention` e che quindi permette il rimpiazzo rapido e senza ulteriori modifiche nei modelli costruiti con `flax.nn.attention`.

In entrambe le funzioni il parametro principale è rappresentato dal numero di proiezioni random da considerare per l'approssimazione della matrice di Attention. Esso deve essere impostato, secondo le indicazioni date nel paper, pari a $dlog(d)$ con d dimensione degli embedding, ma di fatto negli esperimenti riportati nel codice è pari a 64. Nei task affrontati questo parametro è regolato e può essere o 32 o 64.

3.3 BERTX

Il primo modello considerato per l'applicazione del Performer è stato BERT. Come punto di partenza per studiare la libreria FLAX e l'utilizzo del meccanismo di FAVOR+ si è scelta la cartella GitHub contente il codice applicato al dominio specifico delle proteine [5] e citata anche nel paper del Performer.

Questo lavoro, presentato a Luglio del 2020, costituisce la prima applicazione pratica del Performer ad un task. Il codice si inserisce in un ampio studio sulle proteine e sulla loro struttura quaternaria [27] [19]. Dunque gli esempi riportati nel codice sono riferiti a questo contesto preciso di ricerca.

[5] https://github.com/google-research/google-research/tree/bde406c1a1255872f7de4b98763ff81b128ca4b3/protein_lm

Gli esperimenti riportati nel lavoro sono basati sul modello BERT e su una sua implementazione in FLAX. Nello specifico la classe di implementazione del modello è `modules.Transformer` nella quale è stato introdotto un parametro nel costruttore corrispondente al tipo di Attention desiderato da utilizzare nel training. Questa caratteristica permette di utilizzare la funzione di Attention voluta e, date le funzioni descritte in sezione 3.2, rende possibile il passaggio dalla Self-Attention quadratica a quella lineare con una semplice specifica di questo parametro. Inoltre, per il training del modello è presente una classe `FalxModels` che mette a disposizione tutti i metodi per l'addestramento della rete neurale e del suo utilizzo.

La classe che implementa BERT, seguendo una caratteristica del modello, non è specifica per un determinato teak ed è dunque necessario ampliarla utilizzandola come elemento base di una rete che aggiunge un layer finale per riportare il risultato nel contesto voluto. Per esempio, se si vuole effettuare una classificazione, occorre aggiungere un layer finale che mappi con livello denso il risultato del Transformer, pari al numero di parole nel vocabolario, a una dimensione pari numero delle classi desiderate.

Come detto precedentemente, il lavoro è stato pensato per esperimenti sulle proteine e dunque i dati utilizzati sono specifici per questo contesto e non sono pubblici. Le classi, d'altro canto, per come sono state pensate, sono generiche e permettono di utilizzare la medesima implementazione per altri contesti e task. Per raggiungere questo scopo sono stati introdotti i concetti di dominio e vocabolario.

- Vocabolario. La classe `domains.Vocabulary` viene utilizzata per convertire gli input in tokenizer e dunque può differire molto da contesto a contesto. Per crearne uno specifica per un nuovo task è necessario ereditare da essa e definire i metodi per riportare la grandezza del vocabolario e i metodi per effettuare l'encoder e il decoder degli input.

- Dominio. Nel package *domains* sono state implementate diverse classi, come `domains.FixedLenghtDiscreteDomain`, che rispecchiano i vari tipi di dominio possibili. Anche in questo caso, per specificarne uno nuovo occorre ereditare da una di queste classi e apportare le modifiche volute.

Queste due classi rappresentano il contesto e sono specifiche del task sul quale si desidera applicare BERT.

Le caratteristiche riportate sono fondamentali per comprendere le modifiche da apportare al codice e per sfruttare le potenzialità di questa implementazione.

Infine si nota che fra le versioni del modello BERT disponibili, si è deciso di caricare i pesi di BERT Large.

3.4 T5X

Il 21 Dicembre 2020 è stata rilasciata da Google una versione di T5, il cui codice originario era stato rilasciato pochi mesi prima [6], in FLAX [7]. Questa pubblicazione apre all'applicazione del metodo FAVOR+ anche a questo modello. In T5 l'applicazione del Performer risulta ancora più interessante data la dimensione notevole di questo modello, difficile da addestrare altrimenti.

Di seguito si riportano alcuni elementi fondamentali del codice che sono stati ripresi dalla prima implementazione pubblicata che ha lo scopo di implementare e riprodurre gli esperimenti del paper [24] e di offrire del codice per caricare, pre-processare e valutare dataset.

3.4.1 Il Task e le Mixture

L'elemento fondamentale del codice è il Task che lavora con `tf.data.Dataset` ovvero la classe di Tensorflow per i datatset. Un task è costituito da: una sorgente dati, una funzione di processazione del testo, un modello di SentencePiece, una funzione di post-processing, se necessaria, e le funzioni delle metriche da applicare.

In particolare.

- La sorgente dati può essere una qualsiasi funzione che ritorna un oggetto di tipo tf.data.Dataset o un wrapper per dataset scaricabili on-line o letti da file.

- La funzione di processazione dei dati, o text preprocessor, elabora l'input in maniera da riportarlo nella forma text-to-tenx necessaria per poter utilizzate T5.

  ```
  da {'de':'Das ist gut.', 'en':'That is good.'}
  a  {'inputes':'translate German to English: Das ist gut.',
      'targets':'That is good.'}
  ```

  Diversi esempi di funzioni possono essere trovati in `t5.data.preprocessor` ma è possibile definire il proprio.

- Il modello di SentencePiece è utilizzato per trasformare gli input in token e decodificare i token di output.

- Le funzioni delle metriche effettuano il calcolo delle metriche principali per valutare la correttezza dell'output raggiunto dalla rete neurale. Tra

[6] https://github.com/google-research/text-to-text-transfer-transformer
[7] https://github.com/google-research/google-research/tree/master/flax_models/t5x

le metriche implementate ci sono anche: rouge 1/2/L e blueu. Anche in questo caso è possibile implementarne una nuova e specificarla.

- La funzione di post-processing, se specificata, trasforma l'output della tesi dalla versione text-to-text ad un altro formato voluto.

Un esempio di task dunque può essere:

```
TaskRegistry.add(
    "c4_v220_span_corruption",
    TfdsTask,
    tfds_name="c4/en:2.2.0",
    text_preprocessor=functools.partial(
        preprocessors.rekey, key_map={"inputs": None, "targets":
            "text"}),
    token_preprocessor=preprocessors.span_corruption,
    output_features=DEFAULT_OUTPUT_FEATURES,
    metric_fns=[])
```

Un altro elemento fondamentale è rappresentato dalla `Mixture`. Essa rappresenta una classe che può essere utilizzata per raggruppare una serie di task da utilizzare assieme per un addestramento o per un valutazione, come per esempio con il framework GLUE. Questo meccanismo rende il codice molto flessibile e permette, per esempio, di addestrare la rete su un singolo dataset ma di fare il successivo calcolo delle metriche su più dataset. Inoltre permette di fare un addestramento con un dataset che viene composta da input presi da diverse sorgenti, specificandone la percentuale desiderata.

Sia i task e le mixture per essere utilizzati vengono aggiunti in due oggetti Singlethon che fungono da registro e che permettono di recuperarli durante tutta l'esecuzione dell'addestramento.

Il t5.models e l'implementazione FLAX

Nel codice sono presenti dei connettori per collegare i task e le mixture al modello da cui fare il training. Nella implementazione originale della cartella t5 sono presenti il connettore per le versione Tensoflow della libreria Transformer di HugginFace [31].

Inoltre sono definiti dei connettori per addestrare i modelli sul cluster di TPU/GPU di Google o su GPU con PyTorch. A questo scopo è offerta una guida per impostare l'ambiente di addestramento su cluster cloud TPU di Google.

Nell'implementazione FLAX sono stati introdotti gli elementi tipici di questa libreria che permettono di eseguire il codice su diversi device contemporaneamente. Inoltre si sono introdotti diversi parametri di esecuzione che permettono

di configurare l'esecuzione del codice, tra cui: la sorgente dei pesi, url online o file, il checkpointing e la configurazione della topologia del cluster su cui eventualmente eseguire il codice.

Per concludere la descrizione codice si riportano le seguenti note tecniche.

- L'implementazione FLAX non permette il caricamento di pesi di modelli già pre-addestrati provenienti da altre librerie come HugingFace.
- FLAX, utilizzando JIT ha la caratteristica di pre-allocare di default tutta la memoria disponibile e non aumentarla alla necessità. Questa caratteristica può essere gestita e modificata con precise configurazione ma occorre tenerne conto soprattutto se le risorse sono limitate.
- Il codice è strutturato per poter essere eseguito sia con Attention Quadratica che lineare ma non è supportata la parametrizzazione di questa funzione in fase di inizializzazione.

Fra le varie versioni di T5, per gli esperimenti si è utilizzata quella di T5 base.

Capitolo 4

Il contesto del problema

4.1 L'importanza della ricerca nell'ambito medico

Le malattie rare pongono sfide particolari ai pazienti, alle famiglie, ai dottori, ai ricercatori e in generale a tutti coloro che operano in questo ambito. Attualmente sono state scoperte più di 6000 malattie rare e più di 350 milioni di persone convivono con esse, circa il 5% della popolazione mondiale. A causa della scarsa disponibilità di informazioni, nonché della loro dispersione su diverse banche dati, negli ultimi anni si sta assistendo ad una forte crescita nel numero delle comunità di pazienti sui social network come Facebook.

All'interno di queste comunità virtuali, si riuniscono un numero sempre maggiore di pazienti che hanno l'obbiettivo di condividere le proprie esperienze e di trovare delle risposte a quesiti relativi alle proprie condizioni di salute, come ad esempio: "Quali sono i più efficaci e sicuri trattamenti sanitari da punto di vista del paziente?", "Che cosa causa il fallimento di alcuni trattamenti?", "Quali cibi alleviano i sintomi di una determinata patologia?" oppure "Quale centro medico risulta più adatto per trattare le mie condizioni di salute?".

Questi gruppi rappresentano una sorgente importante di informazioni che provengono da esperienze dirette dei pazienti e dunque che pongono il loro punto di vista al centro rendendole più vere e utili per nuove persone affette da malattie o da loro parenti. Inoltre l'analisi di queste informazioni potrebbe rilevare nuove relazioni dirette fra azioni svolte dai singoli pazienti e possibili effetti positivi o negativi sulla loro malattia. I dati che si stanno considerando, però, sono testuali e una loro analisi manuale richiederebbe molto tempo e sarebbe soggetta ad errori umani. Una corrispondente analisi meccanica, attraverso l'uso di tecniche di elaborazione dei dati, potrebbe sintetizzare la conoscenza racchiusa in post di pazienti, come nel caso di Facebook, che si

estende lungo molto anni. I vantaggi sarebbero enormi sia dal punto vista della ricerca medica che esperienziale e di supporto per i pazienti.

4.2 Una nuova metodologia di elaborazione di testi

Nel contesto descritto nella Sezione 4.1 si inserisce il lavoro di ricerca effettuato da Giacomo Frisoni, dottorando presso i gruppo di ricerca NLU del dipartimento di Ingegneria e Scienze Informatiche dell'Università di Bologna, e dei professori Gianluca Moro e Antonella Carbonaro [11].

Il lavoro presentato in questo paper, si pone come obbiettivo l'estrazione di conoscenza da una larga disponibilità di testo non strutturato generato dagli utenti nel corso del tempo, come i post di un gruppo Facebook, in modo tale da presentarlo in modo organizzato e consultabile da una persona interessata all'argomento. Partendo dalla consapevolezza della necessità di integrare differenti metodologie in domini complessi, la ricerca mostra un combinato uso delle tecniche di text mining e semantic web, prendendo come caso di studio l'Acalasia esofagea. In particolare l'output del processo presentato è costituito da ontologie che hanno lo scopo di estendere ORDO [1] e introducono una visione incentrata sul paziente e sulla sua esperienza di vita nel mondo dei linked data.

Il significato di questo sviluppo è che questo potenzialmente costituisce le basi di un progetto in grado di permettere il rapido accesso a molteplici informazioni dall'elevato valore (in argomenti come la sintomatologia, l'epidemiologia, la diagnosi, i trattamenti, i medicinali, la nutrizione e il life style), rispondendo ai quesiti dei pazienti e fornire loro uno strumento ulteriore per supportare il loro processo decisionale, minimizzando i costi attraverso l'automatico recupero di questi dati e l'incremento della produttività dei ricercatori e dei medici. In Sezione 4.3 si illustrano le fasi principali di questa metodologia di estrazione di informazioni.

4.3 POIROT

Il lavoro presentato in tre differenti pubblicazioni [10] [11] [12] si pone l'obbiettivo di indagare le ragioni che spiegano un determinato fenomeno di interesse, identificando correlazioni con una certa significatività statistica. Alcuni esempi di correlazioni generate sono: "agrumi" ↔ "reflusso gastrico": 87% o "GERD" ↔ "patropazolo": 82%. Questo task prende il nome di *descriptive text mining* che è completamente differente dal task di predizione, nel quale

[1] https://bioportal.bioontology.org/ontologies/ORDO

l'obbiettivo è stimare la probabilità di un risultato futuro basato su dati etichettati, come ad esempio la classificazione del testo o la sentiment analysis.

4.3.1 La metodologia

La metodologia proposta, denominata POIROT, è indipendente dal dominio di applicazione e si presta ad operare in due modalità: interattiva e automatica.

Di seguito si descrivo brevemente i passi principali di POIROT [10].

L'elaborazione iniziale dei dati

Come passo iniziale, una pipeline di trasformazione può essere applicata allo scopo di incrementare la qualità dei documenti (come per esempio l'unificazione dell'encoding, la normalizzazione dei simboli, la rimozione degli URL e la gestione di input di lunghezza variabile). Grazie a questa fase un documento come "i sufferer from achalasiaaa" può essere convertito, per esempio, in "I suffer from achalasia". A questo punto dell'analisi una Named Entity Recognition, NER, pre-addestrata può essere usata per effettuare una categorizzazione non supervisionata dei termini contenuti nel corpo, ovvero associare alle parole una determinata entità come luoghi, sintomi, cibi e medicinali. Questa operazione permette di connettere i concetti a basi di conoscenza già esistenti come Wikidata. Inoltre, al dataset di base viene applicata una tokenizzazione dei termini, che permette di incrementare la similarità dei termini e delle loro frequenze, e i dati vengono puliti dalla punteggiatura e dagli spazi bianchi in eccesso. Dopo questi passi preliminari, si definisce il fenomeno da investigare, che viene rappresentato come il modo in cui una determinata classe è distribuita attraverso i documenti. Una classificazione di esempio può coincidere in un task di opinion mining sui dati testuali iniziali e, in questo, la descrizione può avere come oggetto la comprensione delle motivazioni per cui un determinato trattamento medico è fortemente sconsigliato e considerato negativo per i pazienti.

La costruzione del latent semantic space

Successivamente una matrice termini-documenti è estratta dai dati, nella quale ogni riga corrisponde ad un unico termine t, ogni colonna corrisponde ad un unico documento d e ogni cella contiene la frequenza con cui t appare in d. I termini irrilevanti e le stopword possono essere ulteriormente rimosse mantenendo solo i termini con una percentuale di frequenza oltre ad una certa soglia, come ad esempio 1%. Inoltre, per determinare la rilevanza di un termine, in questa procedura, viene applicata una metodologia di assegnazione di un

peso ad ognuno dei termini basata su una variante della classica TF-IDF che fa uso di un fattore inverso all'entropia del termine, come definita da Shannon.

Terminata l'elaborazione iniziale dei dati, si passa all'applicazione di un language model che in questo caso è rappresentato dalla Latent Semantic Analysis, LSA, che presenta i seguenti vantaggi:

- rappresenta un metodo algebrico e le correlazioni semantiche che genera hanno una base matematica e quindi giustificabile e comprensibile;
- permette di mappare sia i termini che i documenti all'interno dello stesso spazio semantico;
- consente di effettuare analisi con un ridotto numero di dimensioni;
- non richiede dati etichettati e un addestramento a differenza dei corrispondenti modelli neurali.

LSA esegue una mappatura del peso della matrice termini-documenti in un spazio vettoriale di dimensionalità minore, chiamato *latent semantic space* che approssima quello originale. La mappatura si basa sul Singular Value Decomposition, SVD, una tecnica di algebra lineare che fattorizza ogni matrice C nel prodotto di separate matrici e riduce la dimensione selezionando i primi k più grandi autovalori singolari e mantenendo così la maggior percentuale di informazione. La matrice generata viene poi visualizzata in termini e documenti nel latent semantic space costruito nel precedente passaggio al fine di identificare le correlazioni tra termini e termini, documenti e documenti e termini e documenti. Inoltre, con questa rappresentazione si può ottenere una risposta grafica sulla distribuzione dei documenti in base alla loro classe, riconoscere l'eventuale presenza di cluster e comprendere l'efficacia dell'applicazione del modello, Figura 4.1.

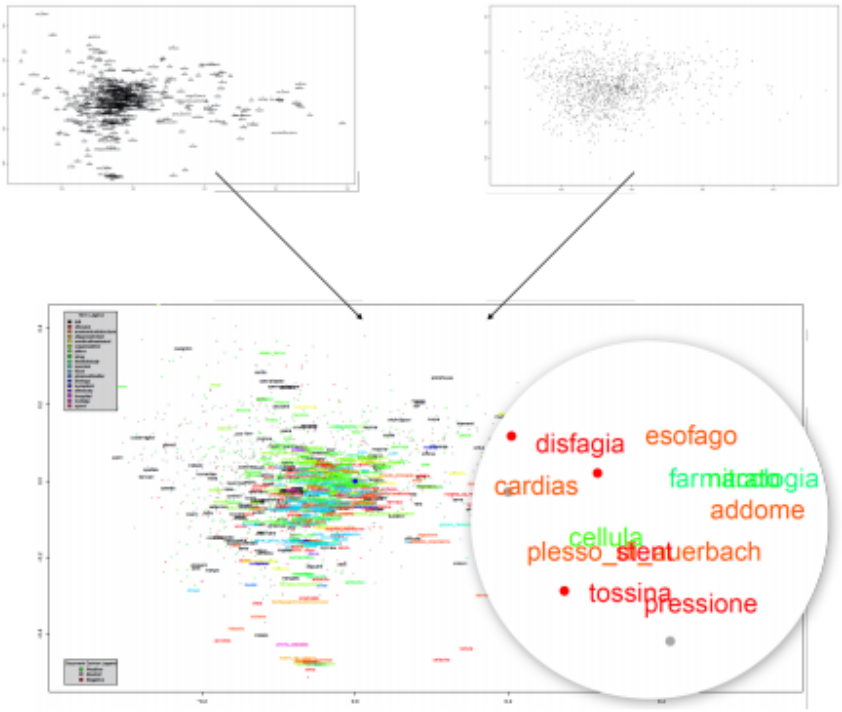


Figura 4.1: Grafo del latent semantic analysis [10]

L'individuazione delle correlazioni

Una volta ottenuto la distribuzione dei dati, si passa all'individuazione delle descrizione dei fenomeni. La metodologia quindi prevede di calcolare le correlazioni termini-termini, documenti-documenti e termini-documenti. Di conseguenza viene introdotta una espansione delle operazioni di LSA, necessarie per l'obbiettivo della ricerca. All'interno della matrice ricostruita di termini-documenti le similarità semantiche tra le coppie di termini o documenti sono misurate con il coseno dei prodotti scalari delle loro rappresentazioni nel latent semantic space. Seguendo la procedura di LSA una query viene mappata nello spazio formato e si calcola la sua similarità con i termini e documenti presenti. Con la metodologia illustrata nel lavoro è possibile costruire una descrizione probabilistica del fenomeno considerato. Per prima cosa, il termine più rappresentativo deve essere identificato visualmente nell'area selezionata con la query voluta. Per dimostrare quindi matematicamente la correlazione tra la query q e la classe c si utilizza il test del chi-quadro si utilizza congiuntamente con R-precision. Più alto è il risultato, minore è la probabilità che l'ipotesi di indipendenza tra q e c e di conseguenza, per ottenere un livello statisticamente significativo associato con la descrizione del fenomeno va considerato il p-value

ottenuto dalla distribuzione del test tra q e c con un certo grado di libertà. In questa maniera si dimostra matematicamente la rilevanza di un termine con la classe voluta e così si può estendere la ricerca ai termini più vicini ad essa. Fra i termini più similari si sceglie il più significativo e lo si inserisce come secondo risultato. Si procede così individuando il numero di termini voluto.

Questa metodologia dunque ritorna un set di termini che non ha una struttura precisa ma sono facilmente interpretabili. I gold standard individuati sono stati successivamente valutati da esperti di dominio che ne determinano la validità da un punto di vista scientifico.

Nel lavoro si presenta anche una variante della metodologia descritta che elimina l'interazione con un operatore e che automatizza tutti i passaggi descritti.

4.3.2 La rappresentazione in un Knowledge Graph

Nel successivo lavoro [12] si utilizzano le correlazioni individuate per popolare un knowledge graph ed estendere quelli già presenti on-line. In particolare si applica un NER ai set di termini individuati, come visibile in Figura 4.2.

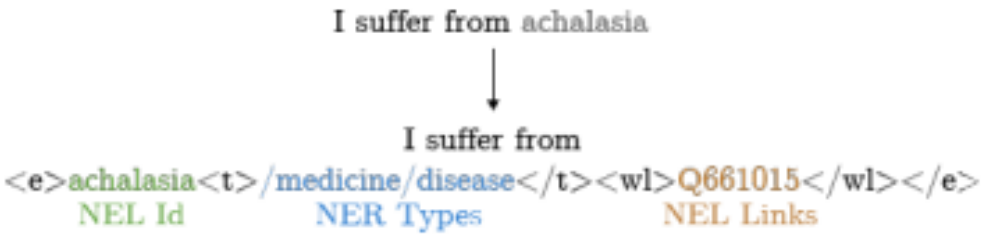


Figura 4.2: Individuazione di entità nei set [12]

L'individuazione di queste entità nei termini viene utilizzata per collegarli ai knowledge graph presenti in rete come DBpedia. Per costruire le relazioni che intercorrono fra i componenti dei set si sono sfruttate le correlazioni statistiche individuate dalla metodologia che vanno a costituire i pesi degli archi dei grafi. Il risultato è dunque simile a quello mostrato in Figura 4.3.

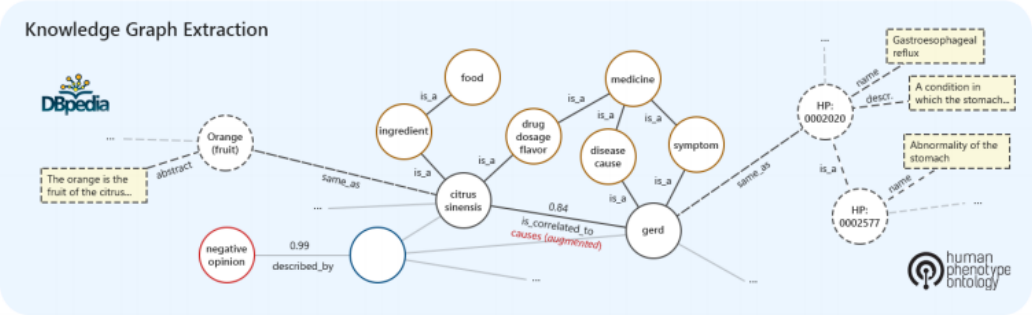


Figura 4.3: Integrazione delle correlazioni con i Knowledge Graph a disposizione [12]

Questa rappresentazione a grafo permette di estendere l'espressività delle relazioni e la aumenta la possibilità di eseguire delle interrogazioni e collegando i termini a delle loro caratteristiche conosciute.

4.3.3 Il dataset

La metodologia POIROT è stata applicata a post su una malattia genetica rara denominata *Acalasia Esofagea*. Essa rappresenta un raro disordine dell'esofago che un'incidenza stimata di 1/10000 ed è caratterizzata da una compromessa abilità di spingere il cibo attraverso lo stomaco, dovuta ad una minore capacità dello sfintere inferiore dell'esofago, LES, di rilassarsi.

Il dataset per l'analisi è stato realizzato con la collaborazione dell'Associazione Malati Acalasia Esofagea, AMAE [2] [3], Onlus, la principale organizzazione di pazienti in Italia per questa malattia. In particolare si è considerato il gruppo Facebook direttamente gestito da AMAE, "Acalasia Esofagea... I malati "rari" non sono soli...!" [4]. Esso colleziona dati dal 2008 ed è composto attualmente da 2000 utenti, tra cui pazienti e dottori. Usando la Facebook Graph API [5], sono stati scaricati 6917 post e 61692 commenti di primo livello, pubblicati dal 21/02/2009 al 05/08/2019. La natura privata del gruppo aumenta la qualità del dataset e limita la presenza di fake news e informazioni non interessanti di altri utenti. Di questo dataset è poi stata realizzata una versione in Inglese. Dall'applicazione della metodologia POIROT sono stati estratte 120 correlazioni positive e 104 negative.

[2] http://www.amae.it/

[3] https://www.orpha.net/consor/cgi-bin/SupportGroupSearch.php?lng=ENdata id=106412

[4] https://www.facebook.com/groups/36705181245/

[5] https://developers.facebook.com/docs/graph-api/

Questi lavoro hanno introdotto una metodo per estrarre conoscenza da dataset testuali non etichettati, ponendo l'attenzione sul punto di vista dei pazienti, dei medici e dei ricercatori. I risultati raggiunti permettono di avere informazioni più precise e sintetiche su una determinata malattia, in questo caso, o sul dominio di applicazione. Il limite principale di questa tecnica è che il risultato è costituito da set di parole a cui deve essere associato un certo significato e una interpretazione, non costituendo delle frasi in italiano complete e corrette. Rappresenta sicuramente un passo in avanti notevole per aiutare le persone ad acquisire informazioni su tematiche così delicate e sconosciute, migliorando per quanto possibile la loro esperienza durante la malattia.

Capitolo 5

I contributi

In questo Capitolo vengono descritte le modifiche apportate al codice di partenza per condurre gli esperimenti voluti.

5.1 L'applicazione dei Performer

La prima fase del lavoro è stata l'introduzione del modello del Performer e del meccanismo FAVOR+ nei modelli BERT e T5. Con questa operazione è stato possibile verificare le performance di questo modello e valutare effettivamente se rappresentasse un passo avanti, considerando anche le risorse limitate non comparabili a quelle di Google.

5.1.1 Il Performer e BERTX

Come primo codice analizzato si è considerata l'implementazione FLAX di BERT [1] dato che ha rappresentato la prima implementazione disponibile e rilasciata dal gruppo di ricerca di Google, si ricorda che T5X è stato pubblicato solo a Dicembre 2020 due mesi dopo.

In questa cartella, come già descritto nella Sezione 3.3, è presente la classe `modules.Transformer` che costituisce il modello da addestrare e presenta un parametro per specificare l'Attention desiderata. Dunque in questo caso, per introdurre il meccanismo FAVOR+ non sono state necessarie delle modifiche importanti. Per comodità si è però introdotta la seguente funzione:

```
def get_performer_attentions(m=64, qkv_dim=None, num_heads=None):
  if (m is None and (qkv_dim is not None and num_heads is not None)):
    d = qkv_dim // num_heads
```

[1] https://github.com/google-research/google-research/tree/bde406c1a1255872f7de4b98763ff81b128ca4b3/protein$_lm$

```
    m = int(d * jnp.log(d))
  return make_fast_softmax_attention(m, lax_scan_unroll=16), \
    make_fast_generalized_attention(m, kernel_fn=jax.nn.relu,
        lax_scan_unroll=16)
```

Essa ha come argomenti la dimensione delle proiezioni da utilizzare per l'approssimazione dell'Attention, ovvero la dimensionalità della matrice low-rank, la dimensione degli array di query, key e value e infine il numero di head dell'Attention. Da questi parametri vengono generate, con le funzioni descritte in Sezione 3.3, le funzioni del kernel SoftMax e di quello ReLU che possono essere passate come parametri ai modelli da addestrare a cui sono stati applicati i parametri voluti.

Questa funzione di utilità è stata utilizzata anche nell'implementazione di T5X.

5.1.2 Il Performer e T5X

Il meccanismo FAVOR+ è stato poi implementato anche nell'implementazione FLAX di T5 [2]. A differenza dell'implementazione di BERTX, nelle configurazioni del modello, non è presente un parametro per la specifica dell'Attention, anche se di fatto il parametro è supportato nel modello interno. Dunque si è resa necessaria l'introduzione del parametro `attention_fn` nella classe di configurazione `TransformerConfig`:

```
TransformerConfig(
      ...
      attention_fn=config.attention_fn,
      ...
      )
```

E dunque, nelle componenti interne del Transformer, invece di specificare l'Attention in maniera statica, si è passato questo parametro dalle configurazioni, come per esempio nel layer della MultiHeadAttention:

```
y = MultiHeadDotProductAttention(
        ...
        attention_fn=cfg.attention_fn)
```

L'introduzione in entrambi i modelli dell'Attention lineare è stata una operazione abbastanza semplice poiché i modelli erano predisposti all'accettazione di un parametro dinamico per l'Attention.

[2] https://github.com/google-research/google-research/tree/master/flax_models/t5x

5.2 Il caricamento di pesi da HuggingFace

L'implementazione specifica dei modelli BERT e T5 in FLAX presenta una struttura che differisce da quella di HuggingFace. Attorno a questa libreria si è formata una vasta di comunità di sviluppatori che addestra i modelli e che pubblica i pesi trovati in un repository pubblico e accessibile, evitando il loro continuo addestramento iniziale. Questi pesi non sono disponibile invece per FLAX e la non corrispondenza della struttura li rende non compatibile con le implementazioni considerate in questo lavoro. Dunque si è resa necessaria la mappatura dei pesi disponibili in HuggingFace con il modello BERTX e T5X.

Questa operazione ha richiesto i seguenti passaggi.

- Analisi delle architetture. Si è reso necessario ispezionare tutti i componenti e i layer delle due rispettive implementazioni HuggingFace e FLAX dei due modelli al fine di determinare la corrispondenza fra essi.
- Realizzazione della mappatura. Per ogni architettura di sono estratti i dizionari corrispondenti ai pesi e si sono effettuate le corrispondenze fra le chiavi di HuggingFace e quelle di FLAX.
- Controllo del processo. Terminata la funzione di caricamento si sono fatti due tipi di controlli. Il primo è costituito dal caricamento dei pesi mappati nel modello in un modello di FLAX e, eseguendo il training, si è verificato che le dimensioni dei nuovi pesi coincidessero o ci fossero delle modifiche da fare. In un secondo momento, in particolare per T5X, si sono confrontati i risultati di predizione disponibili on-line rispetto a quelli ottenuti con il caricamento. Al termine del processo essi corrispondevano dando una prova del corretto funzionamento del caricamento.

Per la conversione è stata presa come riferimento la funzione `convert_from_pytorch` usata in `FlaxBertPreTrainedModel` di HuggingFace [3] e una recente pubblicazione di esempio di importazione dei pesi suggerita dal team di FLAX [4].

La funzione di caricamento pesi risultante è simile alla seguente:

```
def load_weights_from_pytorch(pytorch_model,
                              flax_model,
                              config,
                              debug=False):
  """Load T5-base F32 model weights from HuggingFace PyTorch to
     Google Research FLAX.
```

[3] https://huggingface.co/transformers/_modules/transformers/models/bert/modeling_flax_bert.htmlFlaxBertPreTrainedModel

[4] https://github.com/nikitakit/flax$_b$ert/blob/master/import$_w$eights.py

```
...
flax_model["shared_embedding"]["embedding"] = \
  pytorch_model["shared.weight"].numpy()
...
for i in range(0, num_hidden_layers):

  flax_model["encoder"]["encoderblock_"+str(i)]["LayerNorm_0"]["scale"]
      = \
    pytorch_model["encoder.block."+str(i)+".layer.0.layer_norm.weight"]
    .numpy()
...
return flax_model
```

Questa funzione prende in ingresso due dizionari corrispondenti ai pesi dei due modelli e le configurazioni di questi. In output si ottiene un dizionario che può essere poi caricato in un `optimizer` di FLAX per un modello e successivamente addestrato.

Con questa tecnica è possibile, una volta fatta la mappatura, caricare i pesi di qualsiasi modello in FLAX. In Appendice A si riportano le corrispondenze individuate per i due modelli BERT e T5 fra l'implementazione PyTorch di HuggingFace e quella utilizzata di FLAX.

Nelle Sezioni 5.3 e 5.4 si introducono i cambiamenti apportati all'implementazione T5X.

5.3 Il multi-output

Come già detto per il modello T5 è stato necessario cambiare il codice per introdurre l'Attention modulare. Dunque si è reso necessario inserire la nuova classe nel codice del training. Oltre a questa modifica, è stato introdotto anche il multi-output al fine di migliorare i risultati nei task. Per comprendere i cambiamenti apportati si introduce il meccanismo per la gestione del multi-output BeamSearch utilizzato in T5.

5.3.1 BeamSearch

L'algoritmo di beam search [5] seleziona delle alternative multiple per ogni frase di input ad ogni step temporale con una condizione probabilistica. Il numero B di alternative prodotte dipende sul parametro chiamato beam size o beam width, nell'implementazione utilizzata si è mantenuto il primo nome. Ad ogni step, l'algoritmo seleziona le B migliori alternative con la probabilità più

[5] https://towardsdatascience.com/an-intuitive-explanation-of-beam-search-9b1d744e7a0f

alta di rappresentare le più accurate scelte a quel passo. Questo algoritmo da ottimo risultati ma il suo calcolo è complesso e richiede tempo. Beam search restituisce le più probabili scelte e si differenzia da altri algoritmi che hanno come output una singola sequenza di output. La potenza di questo meccanismo sta proprio nella semplice teoria matematica su cui si basa: la probabilità condizionata. Come per il meccanismo di FAVOR+, se la base teorica è solida ma facilmente spiegabile e comprensibile, è facile comprendere la motivazione dell'output individuato, operazione fondamentale in determinati ambiti.

L'algoritmo

Beam Search, per il primo step di predizione, sceglie B token con la maggior probabilità invece che uno singolo. Successivamente, appende i seguenti B token estratti al gruppo precedente e continua a calcolarne la probabilità della loro esattezza. A questo punto vengono scelti i primi B con valore più alto e si procede in questa maniera per tutti i token. Infine si ritornano le B sequenze che corrispondono alle probabilità complessive più alte.

Di seguito si riporta un esempio per comprendere meglio questo algoritmo. Considerando B=3, VocSize=10000 e come tas la traduzione verso l'inglese.

- Step 1. Si trovano le 3 parole con probabilità più alta date le sequenze di input. Dunque il decoder del Transformer applica la funzione di Softmax alle 10000 parole del vocabolario e si selezionano le 3 parole con la probabilità più alta (come "I" "My" "We") e si salvano le tre parole più probabili.

- Step 2. Si trovano le tre migliori coppie composte dal primo gruppo di parole e dal secondo in base alla loro probabilità. Dunque si considerano le 3 parole dello Step 1 come input del secondo step. Si applica la Softmax alle 10000 parole del vocabolario per trovare le 3 migliori per la seconda parola. A questo punto si eseguono le combinazioni fra il primo gruppo di parole e il secondo e si scelgono le migliori coppie con la probabilità condizionale.

- Step 3. Si trovano le tre mgliori triplette calcolando 30000 (B*dimensione del vocabolario) basandosi sulla sequenza di input e la seconda parola. Per esempio, come mostrato in Figura 5.1 [6] si elimina la frase "My parents" poiché non si trova la prima, seconda e terza parola combinazioni con "My parents" nelle prime 3 combinazioni.

[6] https://towardsdatascience.com/an-intuitive-explanation-of-beam-search-9b1d744e7a0f

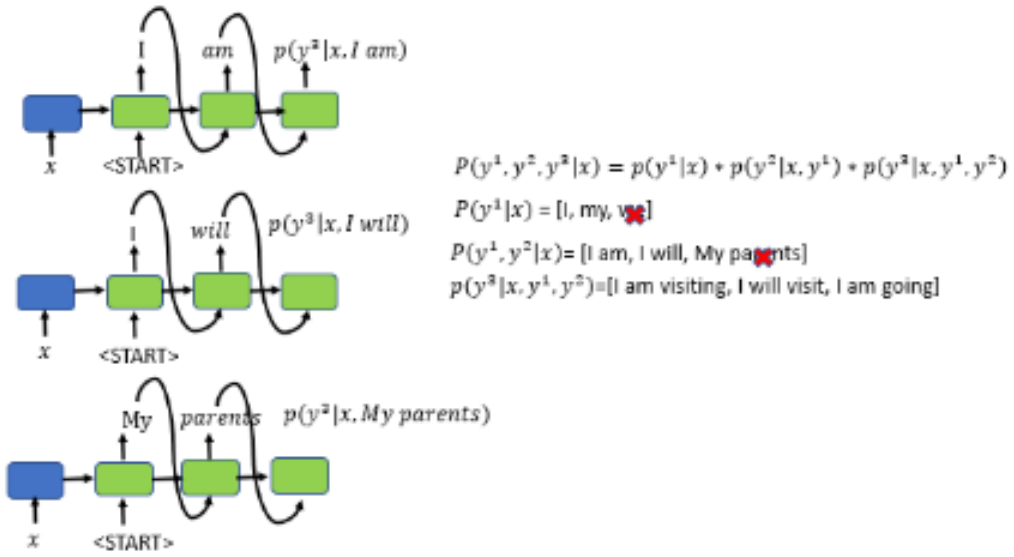


Figura 5.1: Beam search step 3.

Con un parametro più grande di beam size si ottengono traduzioni migliori poiché si cerca sempre più in profondità nell'albero ma si utilizza molta più memoria e risorse computazionali. Con i parametri precedenti per esempio si devono valutare 30000 combinazioni ad ogni step. Questo parametro deve essere dunque un compromesso fra tempi di calcolo e accuratezza della predizione desiderati.

La log-likehood

Con il calcolo delle probabilità condizionali per sequenze lunghe il risultato finale risulta essere un numero molto molto piccolo. Per esempio:
$P(sequence) = p(t1)p(t2)p(t3)...p(t100) = 0.10.10.1...0.1 = 10^(- 100)$
Evidentemente il numero prodotto è molto basso dato che si sta moltiplicando una serie di numeri minori di uno.

Per ovviare a questo problema, in beam search si utilizza la log-likelood. In questo, al posto della moltiplicazione, si sommano i logaritmi delle probabilità.
$log(P(sequence)) = log(p(t1)p(t2)p(t3)...p(t100))$
$= log(p(t1)) + log(p(t2)) + log(p(t3)) + ... + log(p(t100))$
$= log(0.1) + log(0.1) + log(0.1) + ... + log(0.1)$
Dato che il logaritmo di una probabilità ([0,1]) è sempre negativa il valore di log-likehood è sempre negativo. Maggiore è la probabilità e maggiormente la likehood si avvicinerà a zero e quindi sarà più alta. Questa metrica, dato che

per l'algoritmo serve un confronto fra queste probabilità può essere utilizzata al posto della classica probabilità condizionate.

5.3.2 L'implementazione in T5

Per applicare l'algoritmo di beam search è stato necessario modificare principalmente la funzione di `train_lib.predict_step` presente in T5 e la successiva gestione dei nuovi array nel training. In particolare:

```
encoded_inputs = decode.flat_batch_beam_expand(
    models.Transformer(config).apply({'params': params},
                                     inputs,
                                     method=models.Transformer.encode),
    beam_size)
raw_inputs = decode.flat_batch_beam_expand(inputs, beam_size)

beam_seqs, beam_seq_scores = decode.beam_search(
    inputs,
    cache,
    tokens_ids_to_logits,
    beam_size=beam_size,
    alpha=0.6,
    eos_id=eos_id,
    max_decode_len=max_decode_len)


if return_entire_beam:
  # Return the highest scoring beam sequence
  return beam_seqs[:, :, 1:], beam_seq_scores[:, :]
else:
  return beam_seqs[:, -1, 1:], beam_seq_scores[:, -1]
```

Con questa funzione si permette al Transformer di poter gestire una input pari a batch_size * beam_size ad ogni iterazione e, dopo la predizione, si effettua un passo di beam search sulle sequenze di input prese dall'encoder, come descritto nell'algoritmo.

Inoltre, si è effettuata una piccola modifica alla funzione per poter ritornare, oltre che il numero di frasi specificate anche la loro log-likehood, in maniera tale da poter confrontare se la frase che è considerata migliore dall'algoritmo è effettivamente la più convincente anche con una valutazione umana.

Il multi-output è stato aggiunto poiché in task di generazione di frasi è difficile che una singola predizione soddisfi i requisiti imposti. Nei task si è utilizzato come parametro di beam_size 4.

5.4 Il calcolo della ROUGE

Con l'introduzione del multi-output si è dovuta modificare la metrica principale utilizzata per valutare l'accuratezza dei task, la ROUGE. Per comprendere i cambiamenti apportati occorre conoscere le caratteristiche di questa e comprenderne il funzionamento.

5.4.1 La ROUGE

La ROUGE [18], o "Recall-Oriented Understudy for Gisting Evaluation", è un set di metriche per la valutazione di predizioni text-to-text, come quelle di T5 appunto, nel natural language processing. L'algoritmo di score di ROUGE calcola la similarità tra un documento candidato e una collezione di documenti di riferimento ed è utilizzata per valutare la qualità del testo generato. La ROUGE è basata solamente sulla sovrapposizione di testo: può determinare se gli stessi concetti generali sono discussi tra un testo generato in automatico e uno di riferimento ma non può controllare se il risultato sia coerente o meno da un punto di vista semantico.

La ROUGE ha come risultato un scalare con valore compreso tra 0 e 1 o Nan. Un valore alto corrisponde ad una sovrapposizione media di n-gram elevata fra le predizioni e il riferimento. Dunque:

- uno score vicino a 0 indica una similarità molto bassa;
- uno score vicino a 1 indica una similarità molto accentuata, solo se il candidato è uguale al riferimento, il valore è pari a 1;
- se sia il candidato che il riferimento sono vuoti, la ROUGE restituisce un Nan.

La ROUGE comprende differenti tipi di misurazione: la ROUGE-N misura la il numero di sovrapposizioni tra n-gram, per esempio 1 o 2, fra il candidato e il riferimento mentre la ROUGE-L, in cui L sta per "Longest Common Subsequence", misura la più lunga sequenza di parole corrispondenti.

Per il calcolo della effettivo della ROUGE si combinano la Precision e la Recall. La Recall corrisponde alla valutazione di quanto le predizioni generate catturano la frase target originale, sia nel caso in cui la fonte è un uomo sia che essa sia stata prodotta a sua volta da una rete. Per esempio:

- frase originale: "you know nothing john snow"
- frase predetta: "you now nothing little john snow"

- $$Recall = Precision = \frac{\text{numero di parole in comune}}{\text{numero di parole considerate valide}} = \frac{5}{5} = 1$$

La Precision misura quanto la predizione generata sia desiderata o rilevante:

$$Precision = \frac{\text{numero di parole in comune}}{\text{numero di parole generate nella predizione}} = \frac{5}{6}$$

In ROUGE queste due metriche non sono sufficienti e vengono combinate con la F-measure:

$$ROUGEN(recall) = \frac{\sum_{r \in references} \sum_{gram\_n \in r} count\_match(gram\_n)}{\sum_{r \in references} \sum_{gram\_n \in r} count(gram\_n)}$$

Si assuma di voler calcolare ROUGE-2, quindi il match di bigram. Il numeratore $\sum_s$ cicla su tutti i bigram in una singola reference e calcola il numero di volte in cui si trova uno di questi nella frase candidata proposta dal modello. Se ce ne sono più di una, $\sum_r$ si assicura che si esegua questo procedimento per tutte le reference. Il denominatore conta semplicemente il numero totale di bigram in tutte le reference. Questo processo viene poi ripetuto per tutte le predizioni e lo score che viene restituito è la media dei risultati. Per esempio:

- S1: "police killed the gunman"
- S2: "police kill **the gunman**"
- S3: "**the gunman** kill police"

S1 è il riferimento mentre S2 e S3 sono i candidati. Si noti che entrambi hanno una corrispondenza di un bigram con S1, entrambi dunque hanno lo stesso valore di ROUGE-2, anche se S2 è sicuramente migliore. La ROUGE-L gestisce questo caso: in S2 la prima parola e le ultime due coincidono con S1 e dunque possiede uno score di $\frac{3}{4}$ mentre S3 ha solo il bigram e il suo valore corrispondente è $\frac{2}{4}$.

Per i task analizzati si prende in considerazione il calcolo della ROUGE-2 e ROUGE-L.

5.4.2 L'implementazione in T5

L'implementazione originaria di ROUGE implementata dagli autori di T5 [7] è basata sulla `rouge_score` [8] di Google.

[7] https://github.com/google-research/text-to-text-transfer-transformer/blob/master/t5/evaluation/metrics.py

[8] https://pypi.org/project/rouge-score/

Introducendo il multi-output occorre ripensare il meccanismo di calcolo della ROUGE. La metrica e la sua funzione di calcolo rimane ma si modifica il modo in cui le predizioni sono considerate nel calcolo medio. Con, per esempio, quattro predizioni per input, la ROUGE totale viene calcolata come media di un numero notevolmente superiore di coppie riferimento-predizioni, pari a $numerodiinput * beam\_size$. Questo però non rispecchia la motivazione iniziale che ha portato all'introduzione del multi-output: per ogni gruppo considerare solo la migliore e generare più frasi per aumentare la probabilità di individuarla. Per ovviare a questa differenza e migliorare anche il training, si considera, per il calcolo della ROUGE, solo il miglior risultato ottenuto fra le n predizioni trovare per ogni gruppo. In questo modo solo il valore più alto viene considerato e si rispecchia la selezioni che si effettuerebbe manualmente.

A tal fine si è introdotta una modifica al calcolo della ROUGE e si è realizzata la funzione `rouge_top_beam`. In particolare:

```
avg_fmeasure = np.average([score[key].fmeasure for key in score_keys])
if (beam_max_score is None or
  avg_fmeasure > max_avg_fmeasure):
beam_max_score = score
max_avg_fmeasure = avg_fmeasure

seq_idx += 1
if (seq_idx % beam_size == 0):
  aggregator.add_scores(beam_max_score)
  beam_max_score = max_avg_fmeasure = None
```

Con queste poche righe di codice, per ogni gruppo di predizione, si mantiene solo l'oggetto `Score` che si riferisce alla predizione con la migliore F-measure, sfruttando appieno il meccanismo e il significato del multi-output: invece di produrre una singola predizione, se ne ricavano n e di queste si considera la migliore. La nuova implementazione di ROUGE è stata utilizzata per ogni task di T5.

5.5 Il Masked Language Model e la Classificazione su BERT

Come già descritto nella Sezione 5.1.1, il Perfomer è stato applicato, come primo studio, al modello BERT. Questa scelta è stata fatta poiché il modello ha raggiunto delle notevoli prestazioni nell'ambito nel NPL e inoltre, il codice della cartella rappresenta la prima sorgente ufficiale rilasciata da Google per compatibilità con il meccanismo FAVOR+.

IL lavoro svolto su questo modello non ha avuto dunque un obbiettivo di ricerca specifico ma è servito a comprendere il funzionamento di FLAX con i suoi vantaggi e svantaggi e effettuare una verifica parziale delle performance del Performance con delle risorse limitate.

A questo scopo è stato analizzato il task che costituisce la fase di pre-training di BERT, già spiegato ampiamente in Sezione 1.4.2, il Masked Language Model.

Il task è stato utilizzato per valutare quanto i tempi di applicazione del modello con Performer potessero variare in base alla lunghezza della sequenza di input.

Per l'implementazione di questo esperimento si è deciso di caricare i pesi disponibili dalla libreria HuggingFace. Per far corrispondere le due implementazioni, è necessario considerare le seguenti caratteristiche:

- il modello `BertModel` di HuggingFace non è specializzato per nessun task e termina con un layer denso di dimensione numero hidden layer x numero hidden layer, per esempio 768x768;
- HuggingFace offre numerose varianti che estendono il modello base considerando differenti layer finali e le relative funzioni di loss basate sul specifico task desiderato, fra queste di considera il MLM;
- `FlaxBert`, presa dal team di ricerca di Google è preaddestrato solo su task di classificazione, quindi ha un pooler layer con la stessa dimensione del vocabolario.

Con queste premesse, si è resa necessari all'implementazione di una Head, testa, finale in FLAX da aggiungere al modello base per ottenere una piena corrispondenza con il caricamento dei pesi da HuggingFace. Il risultato è stato sviluppato prendendo come riferimento l'implementazione HuggingFace [9].

```
class TransformerWithMLMHead(nn.Module):
    ...
    x = inputs
    x = modules.Transformer(...)
    heads['transformer'] = x
    x = nn.Dense(
        x,
        x.shape[-1],
        kernel_init=nn.initializers.xavier_uniform(),
        bias_init=nn.initializers.normal(stddev=1e-6))
    x = nn.LayerNorm(x)
```

[9] https://huggingface.co/transformers/_modules/transformers/modeling_bert.htmlBertForMaskedLM

```
    heads['prediction_head'] = x
    x = nn.Dense(
        x,
        vocab_size,
        kernel_init=nn.initializers.xavier_uniform(),
        bias_init=nn.initializers.normal(stddev=1e-6))
    heads['logits'] = x
...
```

Dunque è stata realizzata una classe dal nome `TransformerWithMLMHead` che aggiunge al Transformer in FLAX due livelli densi.

La stessa operazione è stata effettuata anche per il task di classificazione.

```
class TransformerWithBinaryClassificationHead(nn.Module):
..
    x = inputs
    x = modules.Transformer(...)
    heads['transformer'] = x
    x = nn.Dense(
        x,
        2,
        kernel_init=nn.initializers.xavier_uniform(),
        bias_init=nn.initializers.normal(stddev=1e-6))
    x = nn.dropout(x, rate=0.3, deterministic=not train)
    x = nn.log_softmax(x)
    heads['logits'] = x
...
```

In questo caso si è aggiunto un solo livello denso che riporti l'output del Transformer al numero di classi desiderate.

Per poter provare il codice su input testuali e non relativi alle proteine si sono poi definiti un vocabolario con tokenizer, `TokenizerVocab` e un dominio, *BertDiscreteDomain*.

Infine si sono uniti tutti gli elementi testando le performance del modello su un input di lunghezza variabile e sempre maggiore. Per effettuare questo esperimento si è realizzata la funzione `test_lm_speed_by_sequence_lenght`. Essa setta la dimensione del dominio al valore voluto, inizializza il modello e esegue una serie di predizioni su un input casuale. Come output viene riportato la media dei valori ottenuti per ogni lunghezza dell'input.

Queste implementazioni hanno permesso di mettere mano al codice del Performer e, con queste prove, si è poi riuscito, oltre ad ottenere dei primi risultati, a lavorare in maniere più spedita sul codice dell'implementazione FLAX di T5.

5.6 Il Fill-in-the-Blank

Il passo successivo all'implementazione dei task di BERT è stato quello di considerare il codice in FLAX di T5. Dopo una prima analisi del codice si è passati allo studio e all'implementazione di un primo task chiamato Fill-in-the-Blank. Questo problema è stato introdotto nel lavoro pubblicato dal gruppo di ricerca di Google e di seguito si riporta una breve descrizione del meccanismo che lo contraddistingue.

5.6.1 Caratteristiche del task

Il problema che è stato considerato dai ricercatori è quello di far apprendere alla rete le relazioni semantiche fra le parole in maniera tale da poterle prevedere con esattezza dato un incipit. In questo tipo di task si lavora con un dataset finale non etichettato e dunque anche i dati di addestramento della rete non le devono possedere. Come già descritto in Sezione 1.4.1, per apprendere questo tipo di conoscenza si può utilizzare il meccanismo del transfer learning [24]. Per implementarlo si utilizzano un approccio di tipo causale. Di recente, come riportato nel paper, però si è recentemente dimostrato che l'utilizzo di una tecnica di denoising delle frase produce migliori performance. L'implementazione di questo meccanismo è data dal Masked Language Model di BERT e proprio a questo si ispira Fill-in-the-Blank di T5. In questo algoritmo il 15% dei token della sequenza di input vengono selezionati e eliminati. Ad ogni spazio creato della frase, anche se corrisponde a più parole consecutive, viene associato un singolo token sentinella a cui poi viene assegnato un token ID che univoco nella sequenza. Questi identificatori sono token speciali che sono aggiunti al vocabolario e non corrispondono a nessun wordpiece. Il target del task dunque corrisponde a tutti gli span dei token eliminati, delimitati dagli stessi token sentinella usati nella sequenza di input più un token sentinella finale che marca la fine della sequenza target. Questa operazione è riassumibile in Figura 5.2.

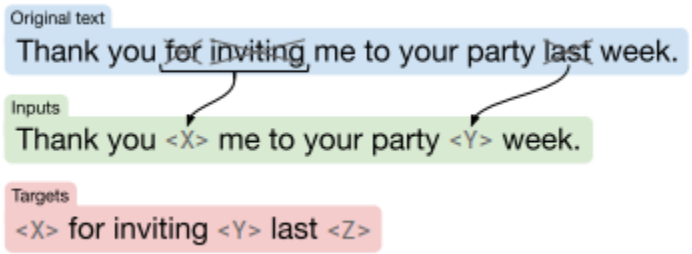


Figura 5.2: L'input del Fill-in-the-Blank [24]

La scelta di mascherare span consecutivi e di predire soltanto i token eliminati è stata fatta per ridurre il costo computazionale del pretraining.

Da questo meccanismo di base è stato presentato un nuovo task che prende il nome di Fill-in-the-Blank, dove il modello al modello è chiesto di rimpiazzare una sezione bianca con un specifico numero di parole o una singola. Per esempio, se viene dato al modello l'input "I love peanut butter and _4_ sandwiches", la rete genera un pezzo di frase composto da 4 parole che completa l'input, quindi "I love peanut butter and **jelly, wihich is what makes good** sandwiches". In generale il numero quattro, nella frase di input, può essere sostituito con un qualsiasi altro numero per addestrare la rete a riempire la frase con un numero diverso di parole.

Questo task si distingue dal Masked Language Model poichè possono essere mascherate più parole consecutive e sostituite con un unico token di mask. Inoltre i token sentinella sono univoci all'interno della frase e non sono rappresentati solo dal token `[MASK]` di BERT. Infine le frasi generate non sono riempite con una singola parola ma i token sentinella possono essere sostituiti da una sequenza di parole volute.

5.6.2 L'implementazione in T5

Per l'implementazione di questo task, come per i successivi, è necessario definire un nuovo task con le seguenti caratteristiche:

- preprocessor: text_preprocessor che analizza un input testuale,
- come dataset di input: delle frasi che vengono caricate come dataset di tensorflow;
- funzione di elaborazione: non è necessario definire nessuna funzione poiché si sta effettuando il task base di T5 che è già implementato nel modello.

Il risultato è il codice seguente:

```
class FillInTheBlankTask(Task):
    ...
    self._split_to_filepattern = split_to_filepattern

    def dataset_fn(split, shuffle_files, seed=None):
      dataset_split = split_to_filepattern[split]
      return dataset_split

    super().__init__(
        name,
        text_preprocessor=text_preprocessor,
```

```
    metric_fns=metric_fns,
    dataset_fn=dataset_fn,
    splits=list(split_to_filepattern.keys()),
    **task_kwargs)
```

5.7 Il task di COMMONGEN

Il secondo problema affrontato nel lavoro di tesi è stato quello di estendere POIROT con un modello in grado di generare frasi dall'output di questa metodologia, ossia il gold standard. Il task dunque si pone l'obbiettivo di generare da un set di parole correlate semanticamente, che possono essere utilizzate per comporre frasi in linguaggio naturale, dell'informazione mancante per un determinato gap conoscitivo producendo in output un fatto veritiero riguardante le parole di input.

5.7.1 COMMONGEN

Il lavoro ha preso come spunto l'elaborato pubblicato il 16 Novembre da un gruppo di ricercatori dell'Università della California e di Washington [17]. Questo lavoro consiste in un task di text generation che, associato ad un novo dataset, addestra un modello nell'abilità di generare del ragionamento di senso comune. In particolare, si vuole ottenere delle frasi coerenti descrivendo uno scenario di tutti i giorni usando i concetti di input.

Il task è molto innovativo e complesso e nelle Sezioni 5.7.1 e 5.7.1 si illustrano dei lavori precedenti che hanno delle similitudini ma permettono di comprendere anche le innovazioni apportate da questo task.

Il task data-to-text

Nel giugno del 2019 è stato presentato un modello in grado di generare del testo da dati specifici come tabelle, fogli excel e altri fonti simili [22]. Il lavoro presentato, applicato al dominio sportivo del football, si pone l'obbiettivo di permutare dati tabellari in un formato testuale che li associ correttamente. Il modello realizzato è innovativo poiché lavora con entità specifiche, argomenti principali del discorso, che non vengono considerate statiche ma ben si dinamiche. Il funzionamento del modello si basa su due elementi fondamentali: la memorizzazione dinamica delle entità e l'Attention gerarchica. Il primo componente permette di aggiornare ad ogni passo di training le informazioni correlate ad una entità, intesta con entità di un NER, e mantenere così il focus per ogni parte del discorso sull'argomento più adatto. L'Attention gerarchica invece, è

un meccanismo di Attention modificato che permette di concentrarsi in una prima fase sulle entità principali della tabella di input e, solo successivamente, sui dati relativi ad esse. Anche se ad un primo sguardo questo task sembra essere simile a quello presentato da COMMONGEN, in realtà esso presenta alcune differenze sostianziali:

- l'input è costituito da tabelle o dati con una struttura precisa e definita, in COMMONGEN l'input sono set di parole;
- l'output è costituito da documenti che possono avere anche lunghezze notevoli, in COMMONGEN è una singola frase;
- al modello è richiesto di associare correttamente le entità ai dati e il testo aggiunto è solo di contorno, in COMMONGEN si cerca di aggiungere delle conoscenza che prima non era data.

Questo lavoro, sebbene non coincida completamente con l'obbiettivo desiderato, riporta alcune informazioni importanti sulla valutazione dei risultati. In particolare, sono stati analizzati 30 documenti con 4 frasi per ognuno di essi da 5 esperti, basandosi sul numero di concetti a supporto o in contraddizione ai dati presenti nell'input. Questa indicazione da una linea guida per una eventuale valutazione umana e sui numeri da poter utilizzare.

Text generation from keywords

Un lavoro datato ma sorprendentemente all'avanguardia è stato pubblicato circa 20 anni fa, [33]. In questa collaborazione fra la New York University e il laboratorio giapponese Communications Research Laboratory si presentano due componenti distinti: un primo modello che, da frasi input seleziona i termini correlati e un secondo modello che da essi produce delle frasi di senso compiuto nella forma di albero di dipendenze. La prima componente prende il nome di *key production model* e può essere equiparata a POIROT, mentre la seconda è denominata *text generation model* e rappresenta l'obbiettivo di questo lavoro di tesi. Il modello presentato per l'estrazione delle parole è probabilistico e considera, appunto, la probabilità che un singolo termine, o nel caso della lingua giapponese presa in considerazione *bunsetsu*, sia una key word con cinque modalità differenti che generano 5 dataset di input diversi. Lo stesso approccio di generare più dataset è stato considerato anche per il task COMMONGEN applicato in ambito medico ed è descritto in Sezione 5.7.2. Anche se il task presenta molti punti in comune con COMMONGEN e con il task considerato nella tesi esso presenta alcune differenze:

- l'apprendimento della struttura della frase è effettuato con regole, approccio oggigiorno superato con l'ausilio del deep lerning;

- i set di input sono estratti da singole frasi mentre con il lavoro svolto è facile acquisirle da un intero corpus;
- la frase di output è minimale e molto breve mentre nel task voluto si cerca di realizzare anche frasi più complesse e con informazioni aggiuntive;
- le frasi di output sono degli alberi di dipendenze e non sono lineari;
- si estrae come output il numero maggiore di relazioni di dipendenza fra le parole di input e questo genera anche molte combinazioni prive di senso.

Si riporta, infine, delle note sulla valutazione applicata all'output. Le frasi generate sono state verificate dagli autori che hanno individuato per ogni output il grado di correttezza morfologica, si sensatezza e di correttezza. Infine è stato valutato se l'output considerato più sensato corrispondesse al primo albero o se fosse presente nei primi 10. Questo principio è lo stesso che ha portato all'introduzione del multi-output nel modello utilizzato per il task.

Caratteristiche di COMMONGEN

Il lavoro di COMMONGEN è composto da due fasi distinte che sono poi state eseguite anche in questa tesi. Nella prima è stato costruito un dataset apposito, attualmente scaricabile [10], partendo da diversi corpora di didascalie. Esse sono state analizzate fino ad ottenere concept set formati da 3,4 o 5 parole che riguardassero il numero maggiore di scenari comuni e che fossero ben equilibrati fra train, evaluation e test set.

Una volta realizzato il dataset, sono state addestrarti diversi modelli, tra cui BERT e T5 in diverse versioni, nel task di predizione di frasi. I risultati poi sono stati valutati con diverse metriche, tra cui ROUGE, e con una valutazione manuale effettuata con diversi volontari.

Il modello risultante dunque possiede una conoscenza sia sulla struttura in generale di una frase in inglese, sia un common sense su scenari di vita quotidiana e riesce a generare descrizioni molto soddisfacenti.

L'importanza di questo task

Il task introdotto da COMMONGEN è molto innovativo e difficile. In particolare occorre sottolineare che è molto complesso in generale generare frasi da concept set che siano corrette grammaticamente e semanticamente, come riportato dai risultati del paper. Questa complessità è data dal fatto che la rete è spinta ad apprendere un ragionamento generativo non semplice da

[10] https://huggingface.co/datasets/common_gen

attuare. D'altro canto questo problema è di particolare interesse per diverse ragioni. Innanzitutto si distingue dai lavori precedenti e anche dai più evoluti modelli in questo ambito. Si consideri GPT3 per esempio. La rete neurale è estremamente precisa nel generare testi da un seed iniziale ma non esiste alcuna dipendenza semantica fra l'input e il continuo. Il testo prodotto non ha dunque alcun vincolo nel significato. Nel task di COMMONGEN le frasi di output devono sia contenere i concetti che costituiscono l'input ma anche correlarli semanticamente e aggiungere informazioni corrette. Infine, la realizzazione di un modello del genere apre la strada a diverse applicazioni in diversi ambiti:

- medico: per aiutare la ricerca medica in determinati casi;
- commerciale: come per esempio in Amazon, per generare recensioni che riassumano le informazioni riportate da tutti i clienti;
- linguistico: realizzazioni di sistemi all'apprendimento di nuove lingue.
- informatico: espansione di training set di frasi etichettate con relativi eventi [37].

In particolare in questa tesi si applica il task di COMMONGEN al modello T5 in ambito medico. L'obbiettivo è quello di ampliare POIROT al fine di produrre come output globale, non semplici set di parole, ma frasi che rappresentino fatti sulla malattia presa in considerazione, la acalasia, e che siano vere per il punto di vista del paziente.

5.7.2 Il dataset

Come riportato nel paper di COMMONGEN [17], gran parte del lavoro è stato la costruzione del dataset di partenza in modo che i set di input fossero formati da parole rilevanti all'intero delle frasi di output. Lo stesso problema si ripropone per l'applicazione del task in campo medico. In particolare, come descritto in Sezione 4.3.3, i dati di base sono costituiti da post Facebook scritti dagli utenti e dalla loro traduzione in Inglese. Da questo dataset di partenze se ne sono realizzati quattro diversi e di seguito si riporta una breve descrizione per ognuno di essi.

Il dataset Fill In The Blank

Per il primo tentativo di generazione delle frase, l'unica elaborazione effettuata è stata l'estrazione di frasi dai Post. Come prima versione si è utilizzato un tool molto semplice che considera divisione di frasi attraverso la punteggiatura. Per il task poi si è sfruttata la funzione di preprocessing di T5X per

il task di Fill in the Blank. Il risultato sono delle frasi con una o più parole mascherate accoppiate al loro originario. Questo approccio però non rispecchia completamento il task di COMMONGEN ed è dunque stato necessario realizzare versioni diverse del dataset.

L'individuazione delle frasi

Al fine di migliorare l'estrazione delle frasi dai post, nelle versioni succeive del dataset, si è utilizzata spaCy [11]. Essa è una libreria gratuita e open-source per il Neural Language Processing avanzato scritta in Python. Fra le sue tanti funzioni si possono trovare: Tokenization, Part-of-speech (POS), Dependency, Parsing, Lemmatization, Named Entity, Recognition (NER), Entity Linking (EL), Similarity, Text Classification, Rule-based Matching, Training e Serialization. Oltre a queste è presente la Sentence Boundary Detection (SBD) che individua e segmenta le frasi all'interno di un determinato testo. La libreria dunque espone diverse pipeline addestrate per l'esecuzione di questi task con supporto in numerevoli lingue.

Per l'estrazione delle frasi dai post Facebook dei pazienti si è dunque applicato il modello addestrato offerto di spaCy per questo task in lingua inglese, `en_core_web_sm`, e lo si è applicato:

```
spacy_model = en_core_web_sm.load()
doc = spacy_model(text)
sentences = [sent.string.strip() for sent in doc.sents]
```

In questo modo, per ogni post di input, si crea una lista corrispondente alle frasi estratte, non pi considerando solo la pnteggiatura, ma un modello addestrato su questo tipo di task.

Il dataset con TF-IDF

Il secondo approccio utilizzato ha come obbiettivo la creazione di un database simile a quello di COMMONGEN in cui si hanno delle coppie formate da un set di termini e dalla loro frase corrispondente. La prima metodologia utilizzata per individuare i set di parole ha considerato la metrica di TF-IDF, "term frequency–inverse document frequency". Questa metrica si utilizza ampiamente nell'elaborazione di testi e indica quanto un termine si importante rispetto ad un documento o ad una collezione di documenti. Il suo valore aumenta proporzionalmente al numero di volte in cui il termine considerato è contenuto nel documento, ma cresce inversamente con la frequenza del termine della collezione. La formula è composta da due componenti:

[11] https://spacy.io

- $tf_{i,j} = \frac{n_{i,j}}{|d_j|}$ in cui $n_{i,j}$ è il numero delle occorrenze del termine i nel documento j, mentre $|d_j|$ è la dimensione documento;
- $idf_i = log_{10} \frac{|D|}{|\{d:i \in d\}|}$ in cui il numeratore è il numero di documenti della collezione e il denominatore è il numero di documenti che contiene il termine i.

Dunque la metrica è calcolata come: $(tf - idf)_{i,j} = tf_{i,j}$ x idf_i. Utilizzando questa metrica come soglia definita come iperparametro, i set e le frasi sono individuati con i seguenti passi per ogni post:

1. si utilizza la metrica di td-idf per indentificare i termini rilevanti del post;
2. si selezionano le frasi nel documento con la libreria spaCy;
3. si selezionano solo le frasi la cui lunghezza è definita da uno range specificato;
4. si considerano solo le frasi i termini che hanno il maggior tf-idf;
5. l'output è dato dai termini con maggior score presenti e le frasi che li contengono. Se più termini sono rilevati nelle frasi allora si generano tutte le combinazioni di essi e si crea una coppia nel dataset per ognuna di queste.

Con questo processo sono state individuate 20498 frasi che contengono almeno tre termini che superano la soglia di tf-idf.

Il dataset con TF-IDF a livello di frase

Calcolando il solo tf-idf a livello di termini si possono scartare delle frasi interessanti. Si consideri una frase con un solo tf-idf con valore molto alto, essa può essere ritenuta interessante per l'analisi ma viene scartata dalla metodologia precedente. Per comprendere anche questa tipologia di input, si è scelto di rigenerare il dataset calcolando la metrica anche a livello d frase. In questo modo si considerano tutte le frasi con il maggior contenuto informativo che, molto probabilmente, saranno anche quelle più interessanti. La tf-idf a livello frase deve essere calcolata come somma dei singoli valori sui termini normalizzata per il numero di parole totali nella frase in maniera da poter confrontare frasi di lunghezza diversa. In questo caso l'algoritmo non cambia di molto da quello precedente: si considerano solo le frasi che hanno una lunghezza compresa nel range stabilito, che hanno un numero di parole compatibile con il numero di elementi minimi per set e che superino un valore minimo di contenuto informativo.

Il dataset con LSA

Nell'ultima variante del dataset si è deciso di individuare il numero di termini con il maggior contenuto informativo attraverso l'utilizzo della LSA. Essa costruisce uno spazio semantico latente che posiziona i vettori relativi ai termini in un nuovo sistema. Effettuando poi la riduzione di dimensionalità low-rank è possibile determinare quali sono i termini con valore più alto. Essi poi vengono considerati come nel metodo precedente e la selezione delle frasi avviene con lo stesso meccanismo. Con gli ultimi due dataset si ottiene un insieme di input che è ben bilanciato sia nel numero di parole nel set, 3 o 4 o 5, che nella lunghezza delle frasi. Per esempio per il dataset generato con LSA si hanno 8326 concept set di lunghezza 3, 6289 di lunghezza 4 e 6098 di lunghezza 5. La lunghezza delle frasi invece è in un range tra 50 e 110 e sono presenti dalle 230 alle 409 frasi per ogni lunghezza possibile. Dunque il dataset è ben bilanciato e queste considerazioni possono essere utili nel determinare la robustezza dei risultati.

La divisione in train, validation e test

L'ultima operazione da eseguire è suddivisione del dataset generato in train, validation e test. Questa suddivisione deve soddisfare alcuni requisiti: i set che corrispondono a combinazioni dello stesso gruppo di parole non devono essere suddivisi fra test e set e le frasi target devono essere suddivise equamente per la lunghezza fra train e test. A questo scopo è stato implementato il seguente algoritmo che utilizza la funzione di sklearn `train_test_split`:

```
train_cs_ids, test_cs_ids = train_test_split(
    commongen_df.concept_set_id.unique(),
    test_size=0.1,
    random_state=42)
train_cs_ids, validation_cs_ids = train_test_split(
    train_cs_ids,
    test_size=0.1,
    random_state=42)
train = commongen_df[commongen_df.concept_set_id.isin(train_cs_ids)]
validation =
    commongen_df[commongen_df.concept_set_id.isin(validation_cs_ids)]
test = commongen_df[commongen_df.concept_set_id.isin(test_cs_ids)]
```

L'algoritmo suddivide gli input in tre gruppi, train, validation e test, elimina, con l'ausilio della funzione `isin`, i set che sono presenti in due divisioni differenti. Il risultato sono tre diversi insiemi che vengono poi salvati su csv pronti per essere utilizzati come input dei task.

5.7.3 L'implementazione in T5

Il dataset appena descritto è stato utilizzato per diversi task basati su di esso e utilizzati sia per testare le performance dell'Attention con kernel SoftMax e ReLU rispetto a quella quadratica, che per valutare la qualità delle frasi generate dai gold standard di POIROT che rappresentano il test set.

Di seguito una breve descrizione dei task implementati.

Il task COMMONGEN

Il primo task realizzato comprende:

- dataset: il dataset del paper di COMMONGEN, scaricato e letto da file;
- preprocessors: una funzione che legge da file una riga e riporta nel formato text-to-text;
- metrica: la funzione di ROUGE progettata per il multi-output.

Il risultato è riportato di seguito.

```
TaskRegistry.add(
  "commongen_task",
  dataset_providers.TextLineTask,
  split_to_filepattern = {
      "train": os.path.join(data_dir, train_file),
      "validation": os.path.join(data_dir, test_file)
  },
  skip_header_lines = 0,
  text_preprocessor = preprocessors.preprocess_tsv,
  metric_fns=[functools.partial(
      rouge_top_beam, beam_size=fine_tuning_cfg.beam_size)] #
          rouge1/2/L
)
```

Si noti che ogni task deve essere registrato nel TaskRegistry con un nome per poi essere richiamato nelle impostazioni.

Il task Acalasia COMMONGEN

Il secondo task, utilizzato per l'effettiva generazione di parole dai post dei pazienti, comprende:

- dataset: il dataset generato dai post nelle varie versioni letto da file;

- preprocessors: una funzione che legge da file una riga e riporta nel formato text-to-text;
- metrica: la funzione di ROUGE progettata per il multi-output.

Il risultato è riportato di seguito.

```
TaskRegistry.remove("achalasia_commongen_task")
TaskRegistry.add(
  "achalasia_commongen_task",
  dataset_providers.TextLineTask,
  split_to_filepattern = {
      "train": os.path.join(data_dir, train_file),
      "validation": os.path.join(data_dir, test_file)
  },
  skip_header_lines = 0,
  text_preprocessor = preprocessors.preprocess_tsv,
  metric_fns=[functools.partial(
      rouge_top_beam, beam_size=fine_tuning_cfg.beam_size)] #
          rouge1/2/L
)
```

La mixture Acalasia COMMONGEN

Infine è stata realizzata anche una Mixture composta dai task precedenti al fine di poter eseguire una doppia valutazione del risultato su due task e dataset differenti nello stesso addestramento. Essa è riportata di seguito:

```
MixtureRegistry.add(
    "commongen_mixture",
    ["achalasia_commongen_task", "commongen_task"],
    0.5)
```

Le impostazioni dei task

Infine per ogni addestramento e utilizzo di un modello occorre specificare una serie di parametri messi a disposizione dall'implementazione. Di seguito si riportano i più imposrtanti.

- mixture_or_task_name e eval_mixture_or_task_name permettono di specificare il task o la mixture per la fase di training e di valutazione;
- save_checkpoints e restore_checkpoints sono dei flag che indicano se occorre salvare i checkpoint o se occorre farne il restore;

- num_epochs, steps_per_epoch e num_eval_steps, come dicono i nomi, corrispondono al numero di epoche, il numero di step di training e quelli di evaluation;
- attention_fn: si utilizza per passare la funzione di Attention da utilizzare nel training;
- use_bfloat16: indica il tipo di numero da utilizzare per salvare i pesi. In particolare questo flag deve essere settato a False atrimenti non vengono supportate le operazioni di esecuzione tra più macchine fisiche.

Con queste impostazioni comune è stato possibile eseguire tutti i task e gli esperimenti desiderati.

Capitolo 6

I risultati raggiunti

6.1 L'implementazione sui server

Lo sviluppo del codice e i primi test sono stati effettuati su Google Colab [1]. Questa piattaforma permette di usufruire delle risorse di Cloud di Google ma presenta alcuni vincoli: le risorse possono essere allocate ad altri utenti senza un avviso, la connessione può durare fino a 12 ore ma può essere interrotta precedentemente e un utente, dopo un prolungato utilizzo delle risorse, può essere bloccato. Queste caratteristiche rendono difficile l'addestramento di modelli per un lungo tempo e rendono necessaria l'individuazione di nuove risorse. Per queste ragioni, gli addestramenti sono stati effettuati su un server messo a disposizione dal Dipartimento di Informatica, Scienza e Ingegneria dell'Università di Bologna. Il server utilizzato presenta le seguenti caratteristiche:

- sistema operativo: Ubuntu 16.04.6 LTS;
- memoria: 1 Tera;
- GPU: due schede TITANX NVIDIA da 12 GB l'una;
- Driver Nvidia: Nvidia-smi 460.27.07, Driver Version 460.27.04, CUDA Version 11.2.

Per realizzare un ambiente per gli esperimenti riproducibile e che possa essere riutilizzato per altri task, si è deciso di utilizzare Docker [2]. Con questa tecnologia è possibile creare dei container dover far girare il codice che lo isolano e permettono di avere più flessibilità sui driver da installare e delle librerie necessarie. Date le caratteristiche dei framework utilizzati si è deciso

[1] https://colab.research.google.com/notebooks/intro.ipynb
[2] https://www.docker.com/

di realizzare due DockerFile, uno per repository e riportati in Appendice B, con le versioni giuste delle principali librerie da utilizzare. In particolare si è deciso di inserire PyTorch, necessario per il caricamento dei pesi, TensorFlow e FLAX, con la relativa versione di JAX, per far eseguire il codice. Questo approccio ha permesso di realizzare una immagine che è poi stata esportata ed è ora disponibile così da poter essere riutilizzata. L'immagine realizzata contiene i principali framework per reti neurali ed è dunque molto versatile e utile. Da essa sono stati poi eseguiti diversi container al bisogno ai quali sono state rese visibile le due GPU disponibili. I container proteggono la macchina da eventuali crash dovuti ad una richiesta di allocazione di memoria superiore a quella presente: in questo caso è il container e fallire e non l'intera macchina che risulta poi inaccessibile.

Con questa immagine infine è stato possibile utilizzare TensorBoard [3], tool basato su TensorFlow che permette di scrivere i risultati degli addestramenti man mano che vengono eseguiti su file appositi. Terminato l'addestramento o durante l'esecuzione, è possibile elaborarli e realizzare in automatico dei grafici che risultano utili per realizzare i report degli esperimenti. Questo tool è stato utilizzato per realizzare i grafici presenti in questo capitolo e in generale per verificare graficamente l'andamento degli addestramenti.

Un altro aspetto tecnico che si è dovuto gestire è stato l'allocazione dinamica della memoria da parte di FLAX e TensorFlow. Entrambi i framework tendono ad allocare tutta la memoria disponibile alla prima istruzione eseguita e quindi bloccando di fatto l'esecuzione. Per ovviare a questo problema sono state settate nei file di esecuzione sulla macchina le seguenti variabile di ambiente:

```
os.environ["TOKENIZERS_PARALLELISM"] = "true"
os.environ["XLA_PYTHON_CLIENT_PREALLOCATE"] = "false"
config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth=True
sess = tf.compat.v1.Session(config=config)
```

Con queste variabile si comunica a FLAX e TensFlow di allocare la memoria man mano è richiesta e questo permette di arrivare alla fien dell'esecuzione senza particolari problemi.

Infine il codice prodotto nel notebook è stato suddiviso in tre tipologie: file contenenti il modello o funzioni comuni, file per l'addestramento dei modelli, uno per task, e file per l'uso dei modelli, anch'essi uno per task.

[3] https://www.tensorflow.org/tensorboard

6.2 Performer vs Transformer in BERT

Il primo esperimento che si è eseguito si è avvalso del codice descritto in Sezione 5.5. L'obbiettivo di questo test è mettere a confronto i tempi di predizione su un input fisso dei modelli del Transformer e del Performer con kernel SoftMax e con kernel ReLU in BERT. In particolare, come riportato nel paper di FAVOR+ [8], il tempo di esecuzione con il nuovo meccanismo e la memoria occupata devono essere minori.

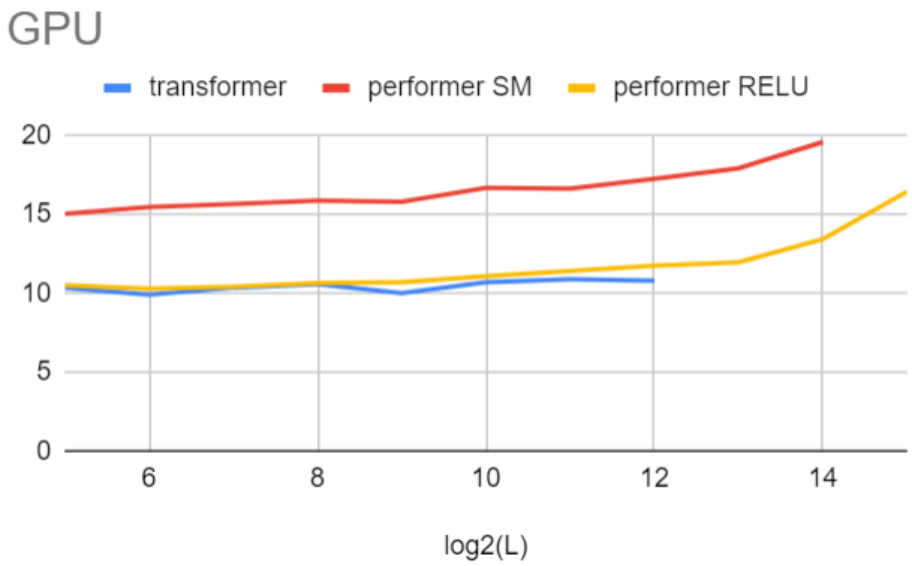


Figura 6.1: Andamento dei Performer e del Transformer in base alla lunghezza dell'input

Dalla Figura 6.1 si può notare come il kernel SoftMax sia notevolmente più più lento rispetto agli altri ma, soprattutto, sia il ReLU che il SoftMax riescono, a parità di risorse allocate, a elaborare lunghezze di input maggiore. Infatti il Transformer con Attention quadratica, identificato dalla linea di colore blu, riesce a portare a termine l'esecuzione senza un problema di allocazione di memoria con una lunghezza dell'embedding pari a 4096. Il Performer SoftMax, anche se più lento, ha come lunghezza massima 16384 mentre il ReLU addirittura una lunghezza di input pari a 32768. Questo esperimento, dati anche i tempi di esecuzione, ha fatto ricadere la scelta della funzione di Attention da usare sul kernel ReLU.

6.3 Le performance del Performer su T5

Una volta stabilito il task finale di generazione di frasi da set di gold standard ricavati da POIROT, chiamato ora in poi Acalasia, e visionando i risultati riportati sul paper di COMMONGEN [17], si è stabilito di concentrarsi sull'implementazione in FLAX di T5 e dunque eseguire gli addestramenti su questo modello. In questa Sezione si riportano gli esperimenti incentrati sulla verifica delle performance di FAVOR+.

6.3.1 La valutazione del Performer al primo caricamento

Il primo esperimento su T5 si è concentrato sulla verifica iniziale del caricamento dei pesi del modello di COMMONGEN. Dunque si è utilizzato il task descritto in Sezione 5.7.3, caricando come persi iniziali di COMMONGEN [4] e facendo una valutazione iniziale della ROUGE sul dataset di test. In particolare, nei pesi che vengono caricati dallp'addestramento di COMMONGEN i valori della ROUGE sono: ROUGE-2 pari a 17.10 e ROUGE-L pari a 39.47. Le prove effettuate sono riportate in Figura 6.2 e corrispondono in ordine alla valutazione iniziale della ROUGE sull'applicazione del modello con: kernel ReLU con m pari a 64, Attention quadratica, kernel ReLU con m pari a 32, kernel ReLU con m pari a 16. Dai risultati riportati si può dedurre che il caricamento dei pesi è stato effettuato nella maniera corretta poiché, con l'Attention quadratica, si raggiungono i medesimi valori pubblicati con i pesi. Inoltre si può notare come l'applicazione di una approssimazione sempre più marcata della matrice di Attention, si passa da m pari a 64 a m pari a 16, inficia di poco sulle performance rilevate. Questo conferma che il meccanismo FAVOR+ diminuisce le risorse utilizzate ma non abbassa la qualità del risultato.

Infine si può notare come l'introduzione del multi-output, Sezione 5.3 e della ROUGE custom, Sezione 5.4, che considera per il calcolo della metrica la frase migliore fra le quattro predette, aumenta notevolmente il valore della ROUGE portandola a 22 per la ROUGE2 e 42 per la ROUGEL, rispetto ai valori 16 e 36 iniziali.

[4] https://huggingface.co/mrm8488/t5-base-finetuned-common_gen

Linear Relu (m=64)

```
100%|██████████| 252/252 [06:45<00:00,  1.61s/it]
INFO:absl:rouge1 => recall = 40.67, 95% confidence [40.14, 41.23]
                    precision = 47.20, 95% confidence [46.65, 47.78]
                    fmeasure = 42.83, 95% confidence [42.33, 43.35]
INFO:absl:rouge2 => recall = 15.38, 95% confidence [14.86, 15.89]
                    precision = 17.74, 95% confidence [17.16, 18.31]
                    fmeasure = 16.14, 95% confidence [15.61, 16.66]
INFO:absl:rougeLsum => recall = 34.69, 95% confidence [34.16, 35.20]
                    precision = 40.07, 95% confidence [39.53, 40.60]
                    fmeasure = 36.48, 95% confidence [35.94, 36.98]
INFO:absl:EVAL eval/commongen_task/rouge1 at step 0: 42.826
INFO:absl:EVAL eval/commongen_task/rouge2 at step 0: 16.139
INFO:absl:EVAL eval/commongen_task/rougeLsum at step 0: 36.482
```

Quadratic Self Attention

```
INFO:absl:Evaluating task commongen_task
100%|██████████| 252/252 [06:44<00:00,  1.61s/it]
INFO:absl:rouge1 => recall = 40.67, 95% confidence [40.12, 41.24]
                    precision = 47.19, 95% confidence [46.66, 47.80]
                    fmeasure = 42.82, 95% confidence [42.30, 43.38]
INFO:absl:rouge2 => recall = 15.37, 95% confidence [14.86, 15.92]
                    precision = 17.73, 95% confidence [17.15, 18.35]
                    fmeasure = 16.13, 95% confidence [15.60, 16.69]
INFO:absl:rougeLsum => recall = 34.69, 95% confidence [34.15, 35.24]
                    precision = 40.05, 95% confidence [39.50, 40.60]
                    fmeasure = 36.46, 95% confidence [35.95, 36.98]
INFO:absl:EVAL eval/commongen_task/rouge1 at step 0: 42.820
INFO:absl:EVAL eval/commongen_task/rouge2 at step 0: 16.131
INFO:absl:EVAL eval/commongen_task/rougeLsum at step 0: 36.459
INFO:absl:BEGIN Train loop.
```

Linear Relu (m=32)

```
100%|██████████| 252/252 [06:47<00:00,  1.62s/it]
INFO:absl:rouge1 => recall = 40.66, 95% confidence [40.07, 41.21]
                    precision = 47.20, 95% confidence [46.58, 47.80]
                    fmeasure = 42.81, 95% confidence [42.24, 43.35]
INFO:absl:rouge2 => recall = 15.37, 95% confidence [14.85, 15.90]
                    precision = 17.72, 95% confidence [17.14, 18.31]
                    fmeasure = 16.12, 95% confidence [15.59, 16.67]
INFO:absl:rougeLsum => recall = 34.71, 95% confidence [34.11, 35.31]
                    precision = 40.08, 95% confidence [39.50, 40.65]
                    fmeasure = 36.49, 95% confidence [35.93, 37.03]
INFO:absl:EVAL eval/commongen_task/rouge1 at step 0: 42.814
INFO:absl:EVAL eval/commongen_task/rouge2 at step 0: 16.119
INFO:absl:EVAL eval/commongen_task/rougeLsum at step 0: 36.488
```

Linear Relu (m=16)

```
100%|██████████| 252/252 [07:06<00:00,  1.69s/it]
INFO:absl:rouge1 => recall = 40.68, 95% confidence [40.11, 41.24]
                    precision = 47.20, 95% confidence [46.61, 47.81]
                    fmeasure = 42.82, 95% confidence [42.31, 43.40]
INFO:absl:rouge2 => recall = 15.37, 95% confidence [14.86, 15.88]
                    precision = 17.72, 95% confidence [17.15, 18.32]
                    fmeasure = 16.13, 95% confidence [15.59, 16.65]
INFO:absl:rougeLsum => recall = 34.71, 95% confidence [34.12, 35.27]
                    precision = 40.07, 95% confidence [39.51, 40.65]
                    fmeasure = 36.48, 95% confidence [35.92, 37.02]
INFO:absl:EVAL eval/commongen_task/rouge1 at step 0: 42.820
INFO:absl:EVAL eval/commongen_task/rouge2 at step 0: 16.126
INFO:absl:EVAL eval/commongen_task/rougeLsum at step 0: 36.476
```

Linear RELU (m=64), BeamSize=4, RougeTopBeam

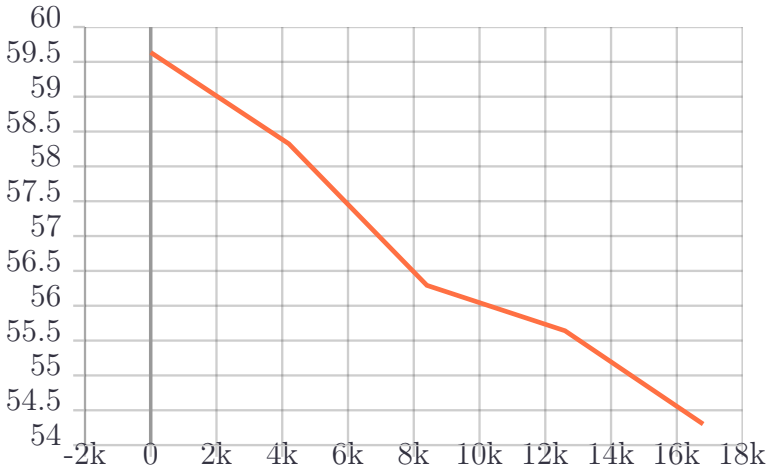
```
INFO:absl:rouge1 => recall = 46.33, 95% confidence [45.74, 46.90]
                    precision = 54.15, 95% confidence [53.50, 54.76]
                    fmeasure = 49.00, 95% confidence [48.40, 49.52]
INFO:absl:rouge2 => recall = 20.98, 95% confidence [20.38, 21.57]
                    precision = 24.42, 95% confidence [23.74, 25.07]
                    fmeasure = 22.11, 95% confidence [21.49, 22.71]
INFO:absl:rougeLsum => recall = 40.22, 95% confidence [39.64, 40.81]
                    precision = 46.75, 95% confidence [46.15, 47.36]
                    fmeasure = 42.44, 95% confidence [41.87, 43.01]
INFO:absl:EVAL eval/commongen_task/rouge1 at step 0: 48.999
INFO:absl:EVAL eval/commongen_task/rouge2 at step 0: 22.106
INFO:absl:EVAL eval/commongen_task/rougeLsum at step 0: 42.444
```

Figura 6.2: Valori di ROUGE per il caricamento iniziale dei pesi di COMMONGEN.

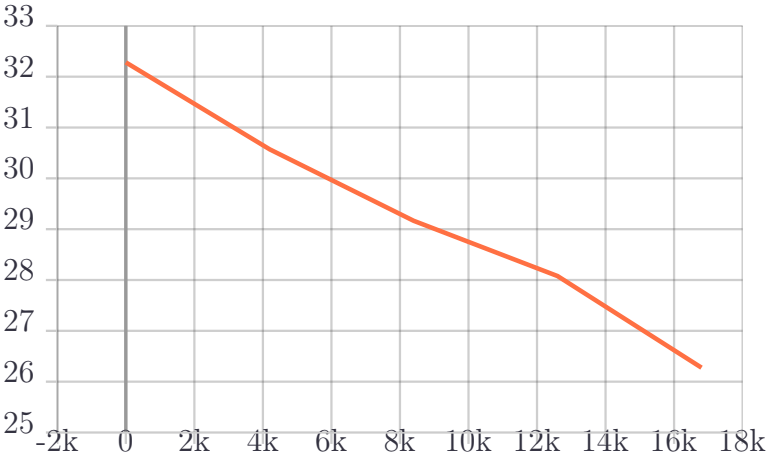
6.3.2 Il Fill-in-the-blank: stabilità dei pesi su dataset COMMONGEN

La seconda prova effettuata sul task di COMMONGEN, comprende la verifica della stabilità dei pesi nel corso di un addestramento. L'obbiettivo è verificare che le prestazione del Performer riportino lo stesso andamento di quelle del Transformer nel tempo. In particolare si è effettuato un addestramento con i pesi di `t5-base-commongen` e con il dataset di COMMONGEN [5] su 5 epoche e 12000 step di addestramento per ognuna. Il task eseguito è descritto in Sezione 5.6.2. In Figura 6.3 si possono visualizzare i grafici realizzati con l'ausilio di TensorBoard inerenti all'addestramento di T5 Base con Attention quadratica con il task descritto. Come è visibile in Figura 6.3 il valore della ROUGE subisce un crollo nel corso delle epoche passando dai valori di ROUGE1=59.64 e ROUGE2=32.28 e ROUGEL=53.16 a ROUGE1=54.3 e ROUGE2=26.28 e ROUGEL=48.51.

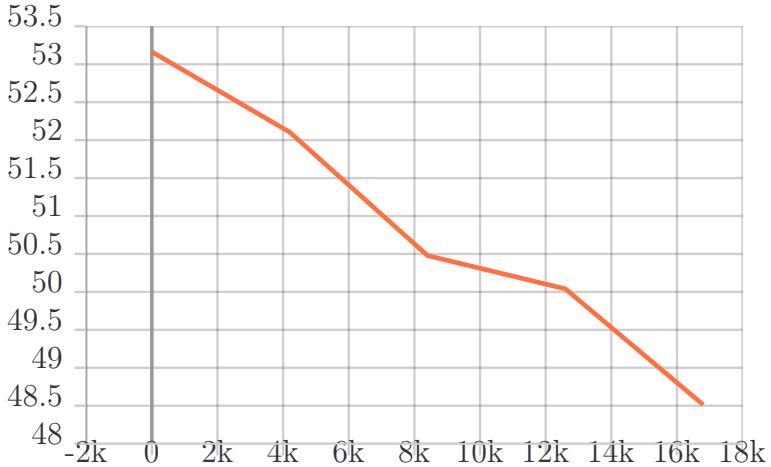
[5] https://huggingface.co/datasets/common_gen



(a)



(b)



(c)

Figura 6.3: Risultati con Attention quadratica: (a) ROUGE1 (b) ROUGE2 (c) ROUGEL

Inoltre si riportano:

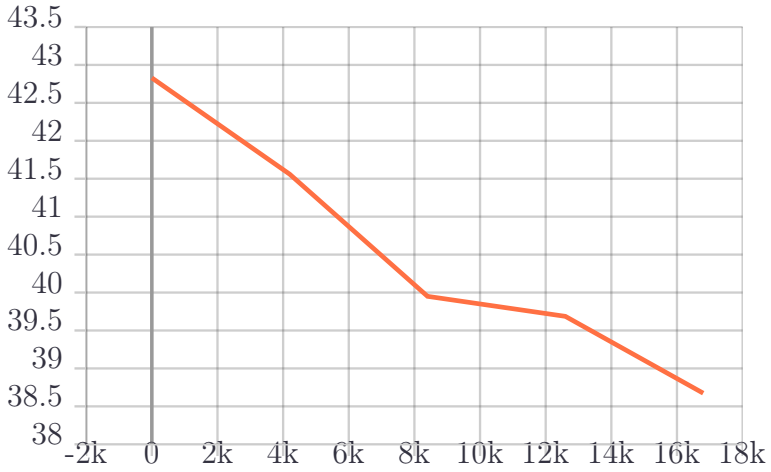
- in Figura 6.4 i grafici inerenti all'applicazione del kernel ReLU con m pari a 64 e beam size 1;
- in Figura 6.5 i grafici all'applicazione del kernel ReLU con m pari a 64 e beam size 4;
- in Figura 6.6 i grafici inerenti all'applicazione del kernel ReLU con m pari a 32 e beam size 4.

Dalla somiglianza delle curve dei grafici si può dedurre che, anche se la ROUGE peggiora nel corso delle epoche, l'andamento è il medesimo sia con l'Attention quadratica che con il kernel ReLU con m sempre minori. Questo conferma ancora una volta che l'applicazione del meccanismo FAVOR+ non peggiora le performance complessive dell'addestramento rispetto all'Attention lineare.

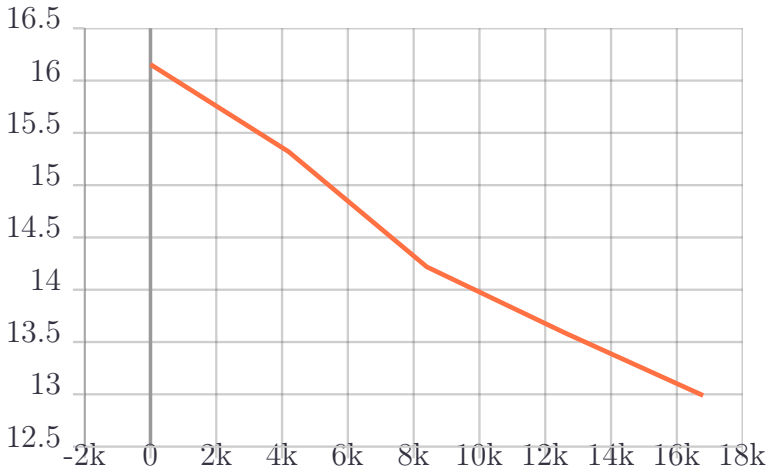
Infine nella Tabella 6.3.2 si riportano i tempi di esecuzione degli esperimenti:

Attention	tempo
Quadratica	7H 16M 23S
ReLU Beam size 1 m=64	7H 19M 30S
ReLU Beam size 4 m=64	7H 30M 13S
ReLU Beam size 4 m=32	7H 21M 38S

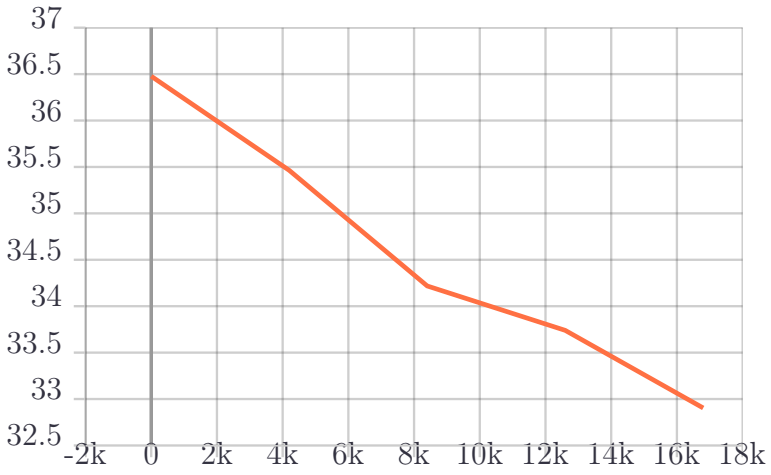
L'analisi dei tempi ci mostra come l'applicazione dell'Attention di fatto non aumenta né diminuisce i tempi ma essi rimangono in linea con le tempistiche del Trasformer. In questo caso non si è riuscito a verificare la proprietà del FAVOR+ di diminuire le tempistiche di addestramento: questo comportamento può essere spiegato dal fatto che l'input preso in considerazione per l'addestramento non ha una lunghezza elevata mentre questo modello presenta le migliori prestazioni con input di lunghezza notevole.



(a)

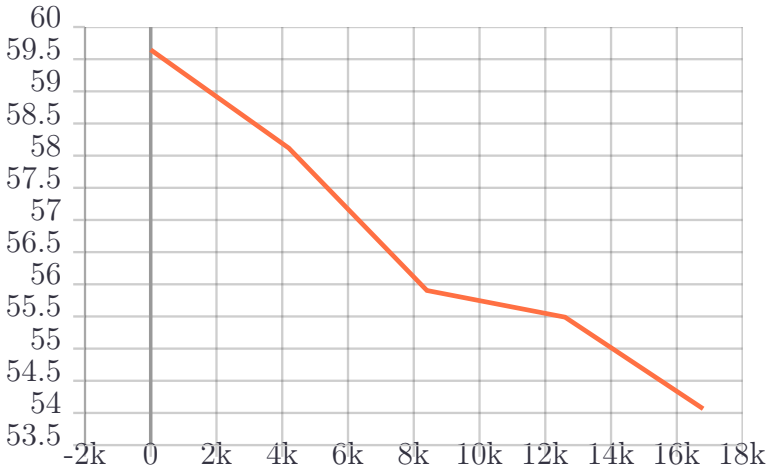


(b)

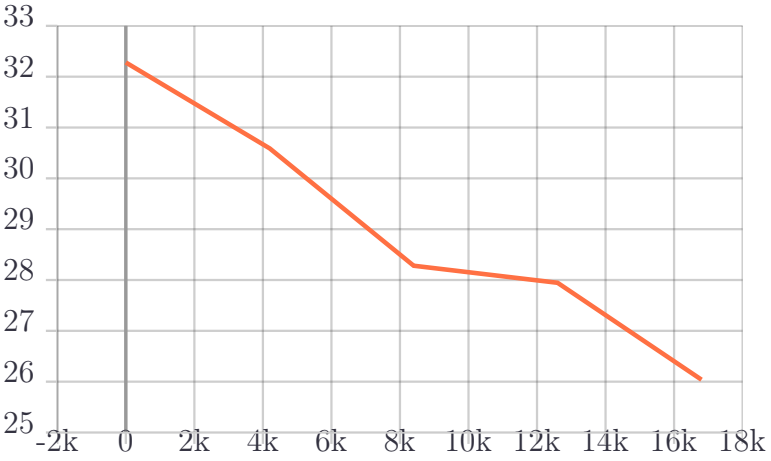


(c)

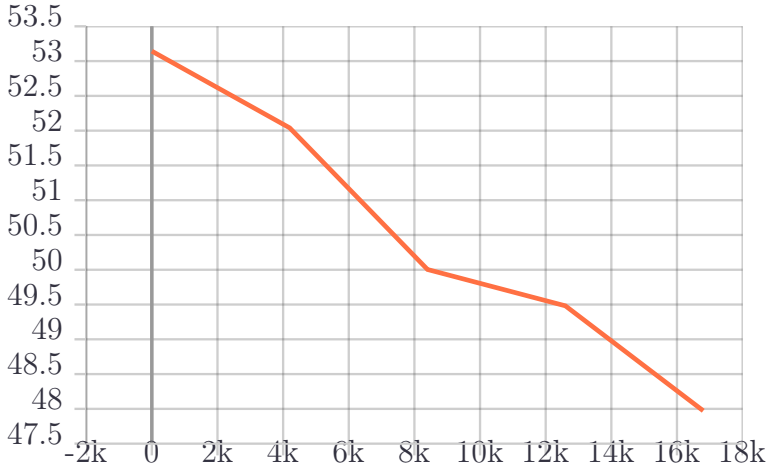
Figura 6.4: Risultati con kernel ReLu e beam size pari a 1: (a) ROUGE1 (b) ROUGE2 (c) ROUGEL



(a)

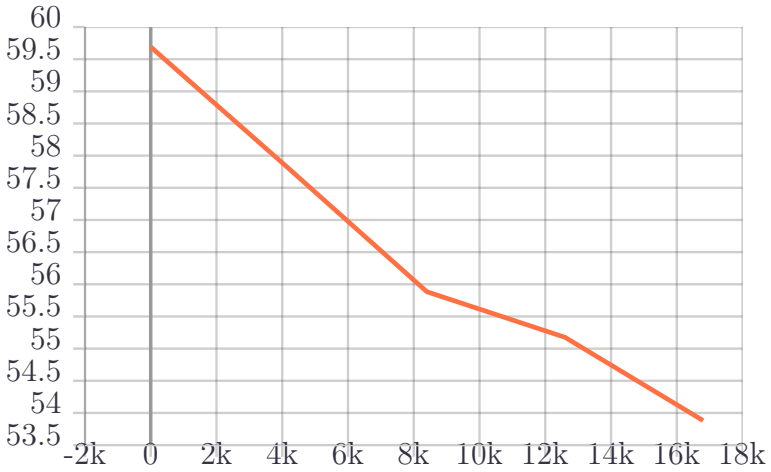


(b)

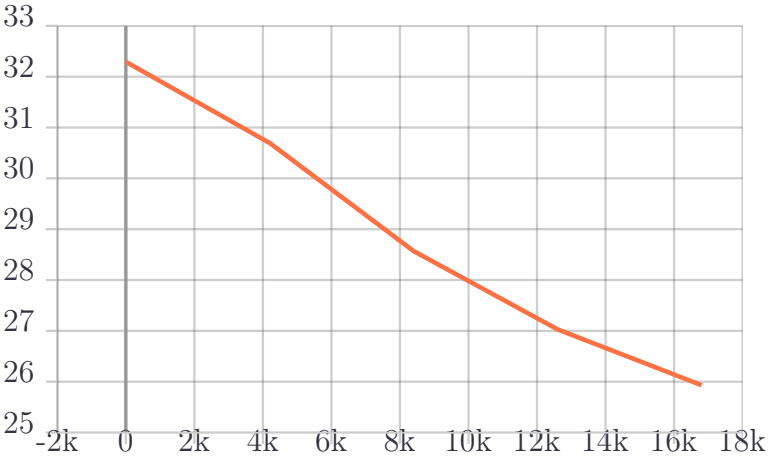


(c)

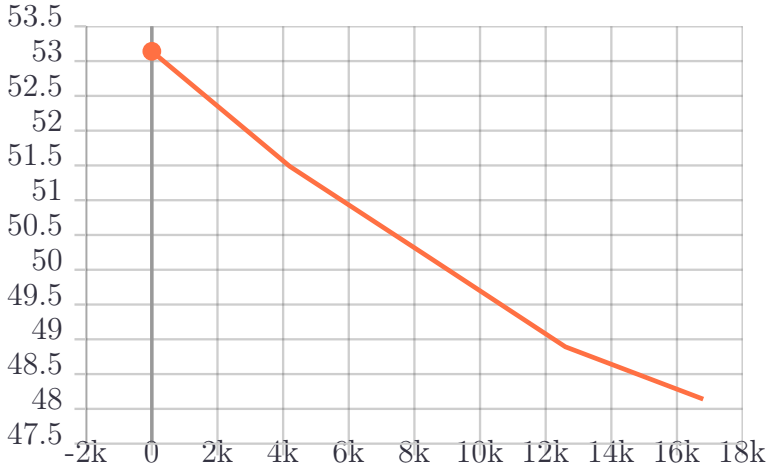
Figura 6.5: Risultati con kernel ReLu e beam size pari a 4 e m pari a 64: (a) ROUGE1 (b) ROUGE2 (c) ROUGEL



(a)



(b)



(c)

Figura 6.6: Risultati con kernel ReLu e beam size pari a 4 e m pari a 32: (a) ROUGE1 (b) ROUGE2 (c) ROUGEL

6.4 Le Performance addestramenti sul dataset Acalasia

La seconda fase degli esperimenti si è concentrata sull'addestramento di un modello che riporti il problema di COMMONGEN nel dominio medico e, in particolare, applicandolo ai post facebook relativi alla malattia acalasia, come descritto in Sezione 4.3.3. Per raggiungere l'obbiettivo definito si sono effettuate due tipologie di analisi. Durante l'addestramento si è calcolata e riportata la ROUGE su due dataset diversi: quello di COMMONGEN e quello di Acalasia realizzato in diverse versione. Una volta ottenuti i modelli si sono considerati i gold standard di POIROT e si sono fatte predire dal modello le frasi corrispondenti. Su queste ultime è stata poi effettuata una valutazione manuale.

Per la prima analisi si è utilizzato come dataset di addestramento quello di acalasia generato come descritto in Sezione 5.7.2 e per l'addestramento si è usata la Mixture riportata in Sezione 5.7.3 con 875 passi per ognuna delle 5 epoche.

In questa Sezione si riportano i valori della ROUGE durante i gli addestramenti. Da questi esperimenti ci si aspetta che la ROUGE sul dataset acalasia aumenti progressivamente mentre quella su COMMONGEN cali. Questo mostrerebbe come il modello apprende conoscenza man mano sul nuovo dominio ma perde invece informazioni sulle quelle iniziali.

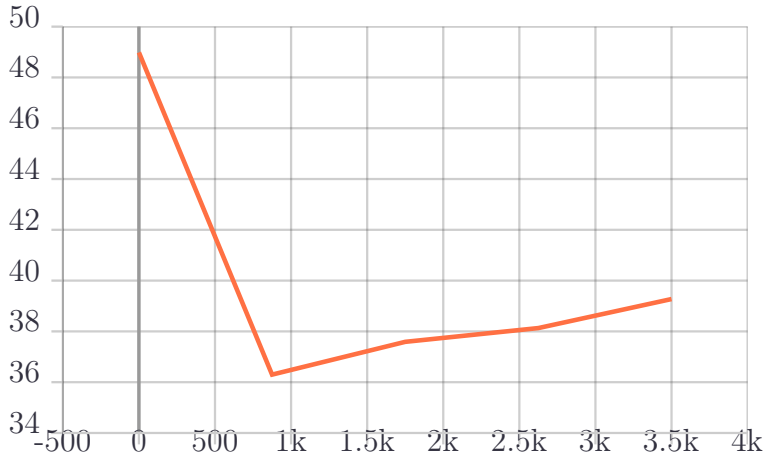
6.4.1 Il dataset con TF-IDF

Il primo esperimento è stato effettuato addestrando il modello sul dataset in cui termini sono estratti con una soglia minima di TF-IDF. Su questo corpus sono state testate 3 differenti Attention: quadratica, kernel ReLU con m pari a 64 e kernel ReLU con m pari a 32.

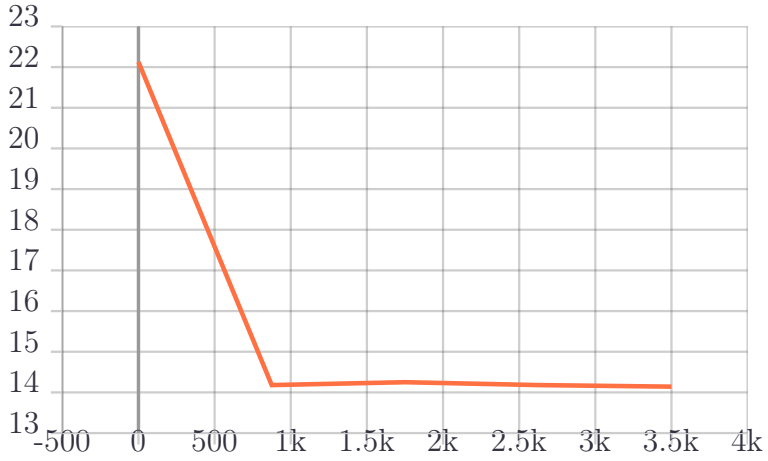
La ROUGE più elevata per tutte le valutazioni sul dataset acalasia, si ottiene alla seconda epoca e il valore massimo si è raggiunto con ReLU m pari a 64 e m pari a 32 con beam size pari a 4: 36.53. ROUGE2, e 54.44, ROUGEL. In generale si sono raggiunti valori molto più alti di quelli riportati nel paper di COMMONGEN [17] e questo può essere spiegato poichè il dominio medico scelto è molto più limitato rispetto al set di argomenti comuni di quel dataset.

Come previsto, inoltre, la ROUGE sul dataset di COMMONGEN diminuisce nelle varie epoche

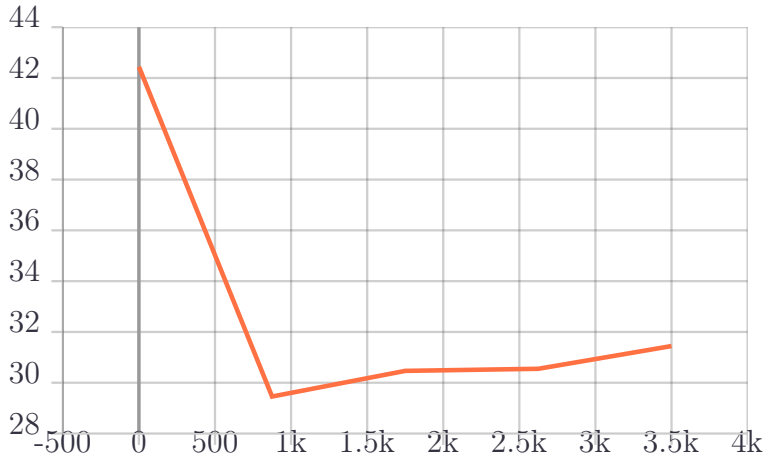
In questo caso si è notato che il tempo di esecuzione globale per l'Attention con kernel ReLU è leggermente inferiore a quello con l'Attention quadratica, 3H 10M 19S contro 3H 31M 3S, confermando questa volta la diminuzione nei tempi di esecuzione con il Performer.



(a)

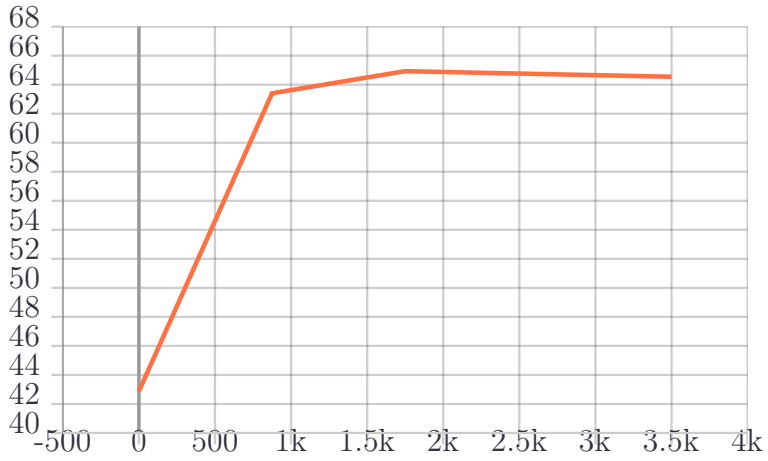


(b)

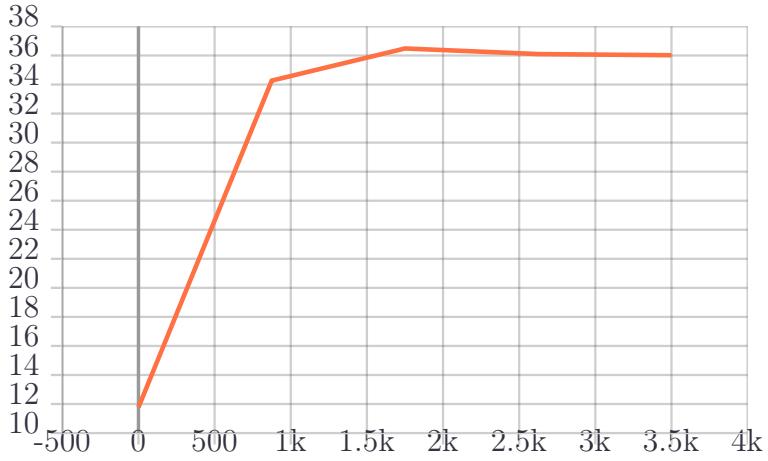


(c)

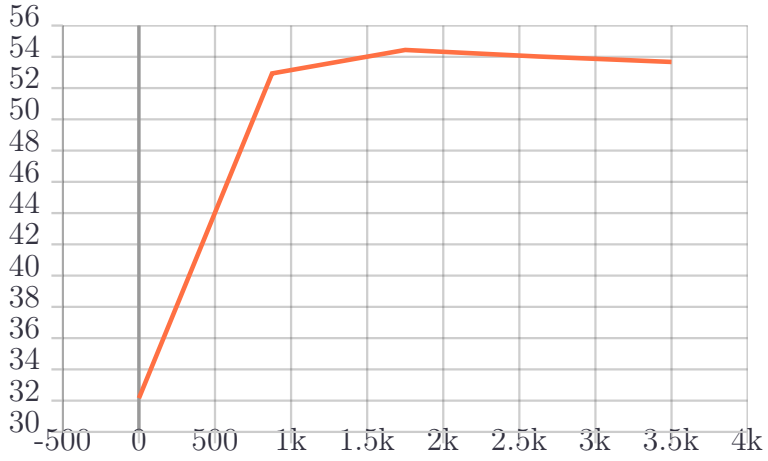
Figura 6.7: Risultati valutazione con Attention quadratica su dataset COMMONGEN: (a) ROUGE1 (b) ROUGE2 (c) ROUGEL



(a)

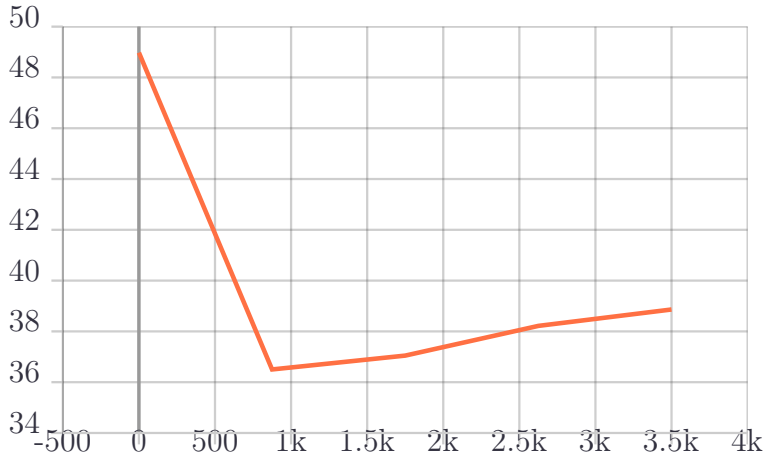


(b)

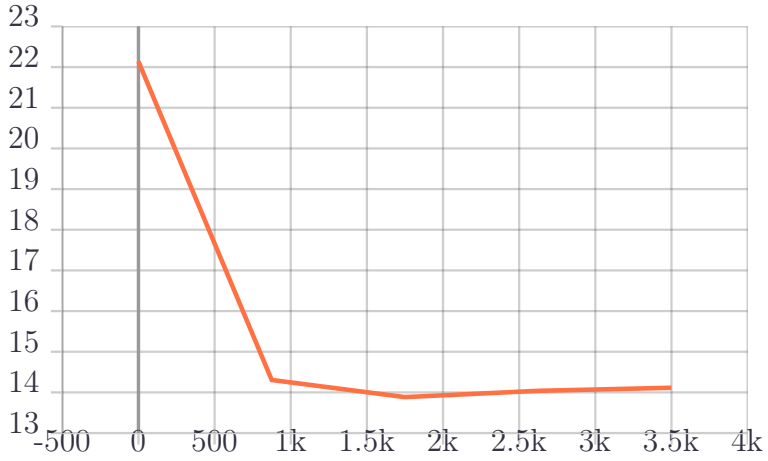


(c)

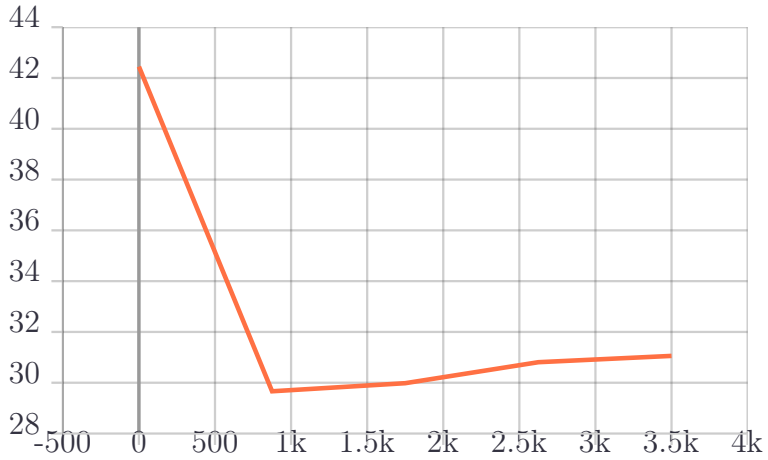
Figura 6.8: Risultati valutazione con Attention quadratica su dataset acalasia con TF-IDF: (a) ROUGE1 (b) ROUGE2 (c) ROUGEL



(a)

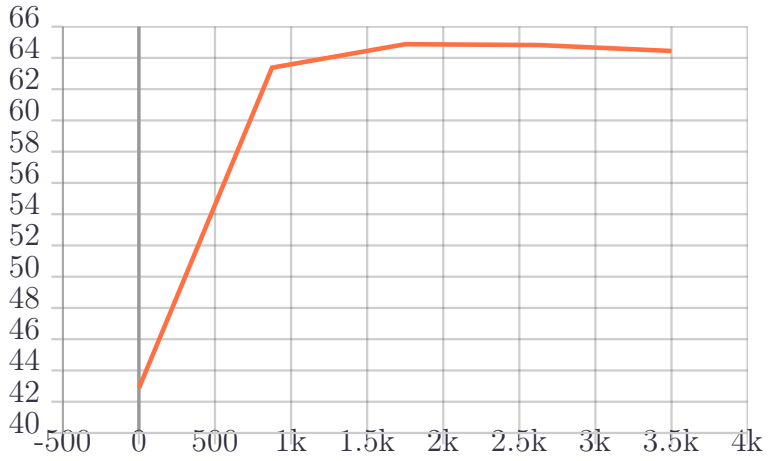


(b)

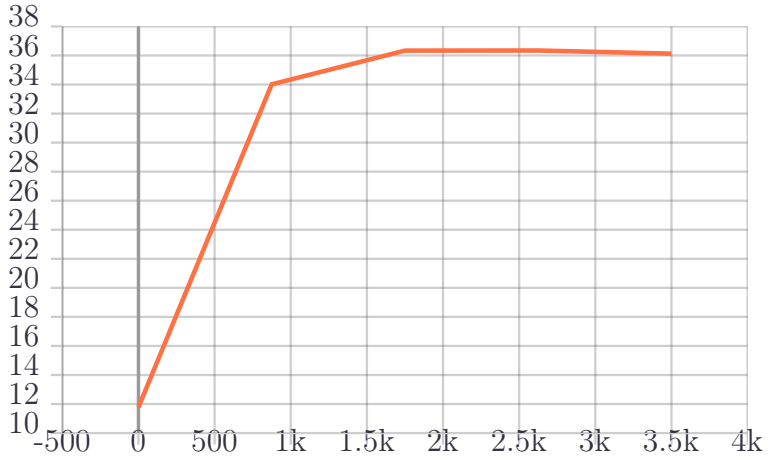


(c)

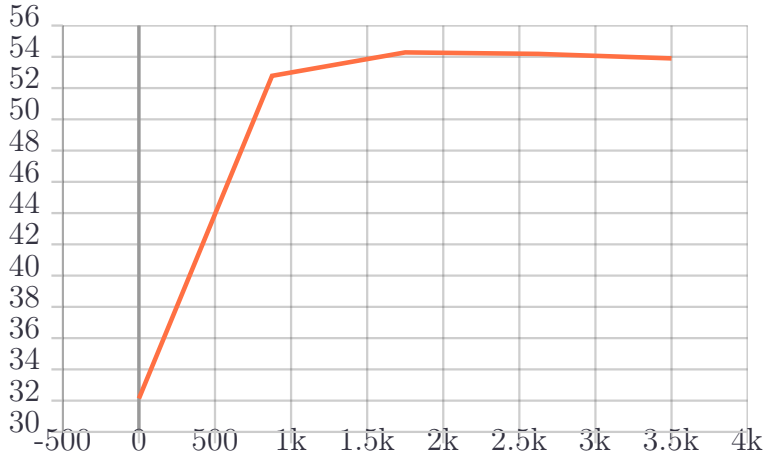
Figura 6.9: Risultati valutazione ROUGE con ReLU su dataset COMMONGEN: (a) ROUGE1 (b) ROUGE2 (c) ROUGEL



(a)

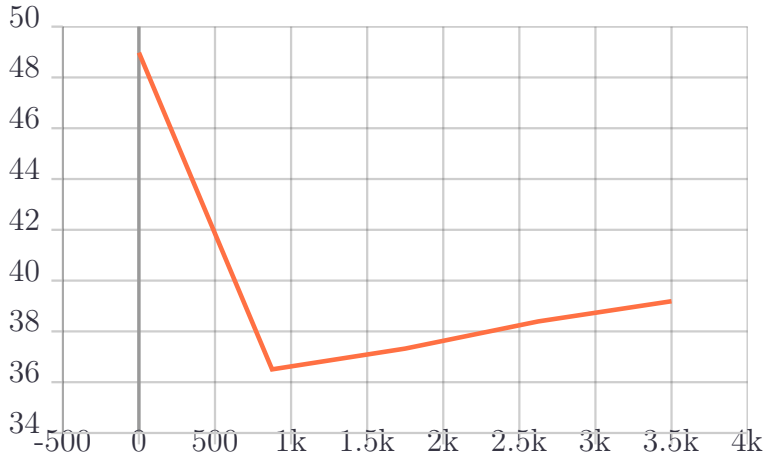


(b)

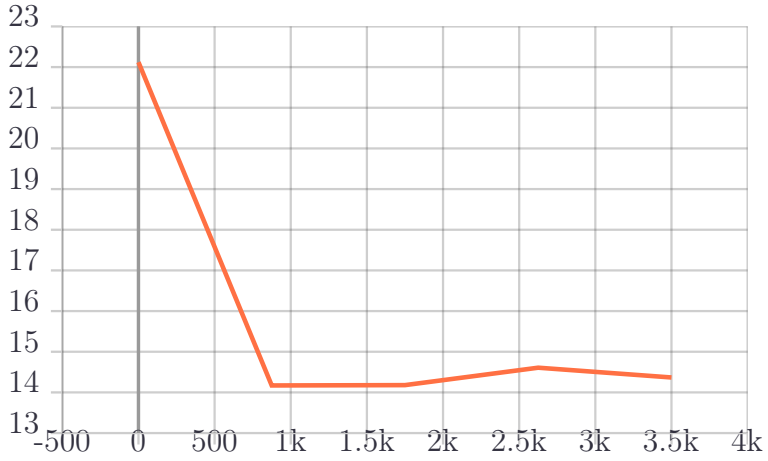


(c)

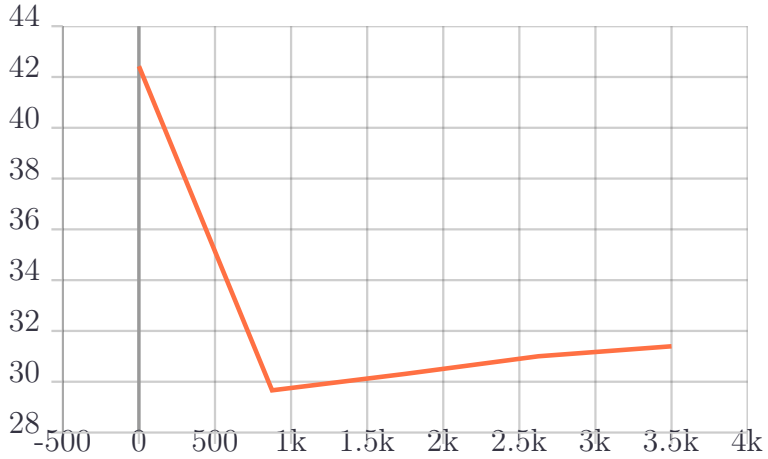
Figura 6.10: Risultati valutazione ROUGE con ReLU su dataset acalasia con TF-IDF: (a) ROUGE1 (b) ROUGE2 (c) ROUGEL



(a)

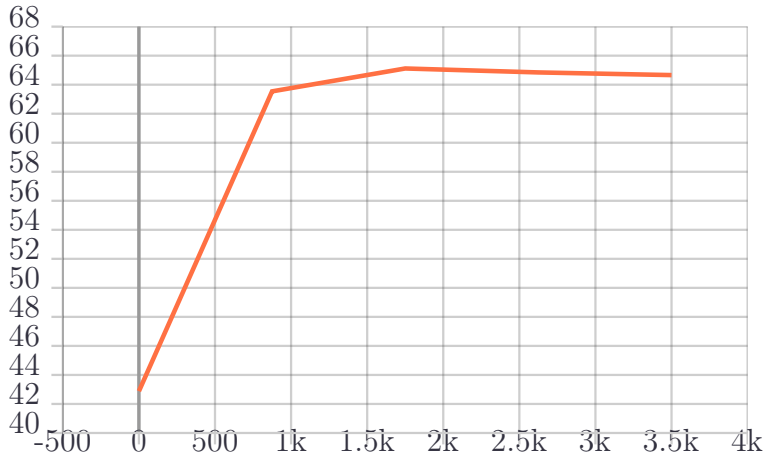


(b)

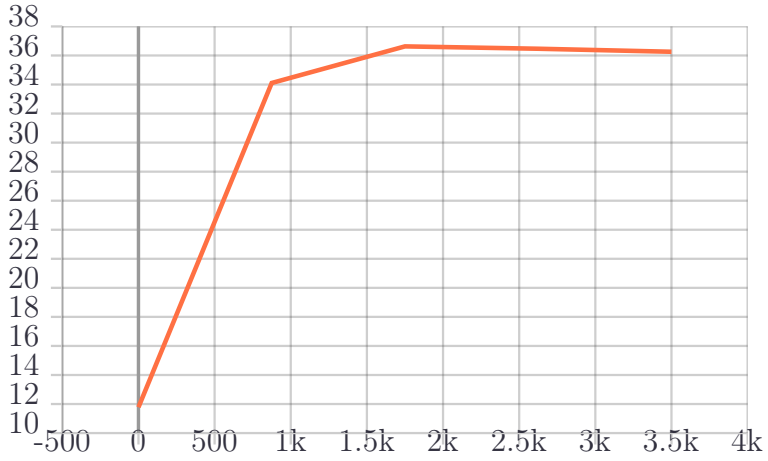


(c)

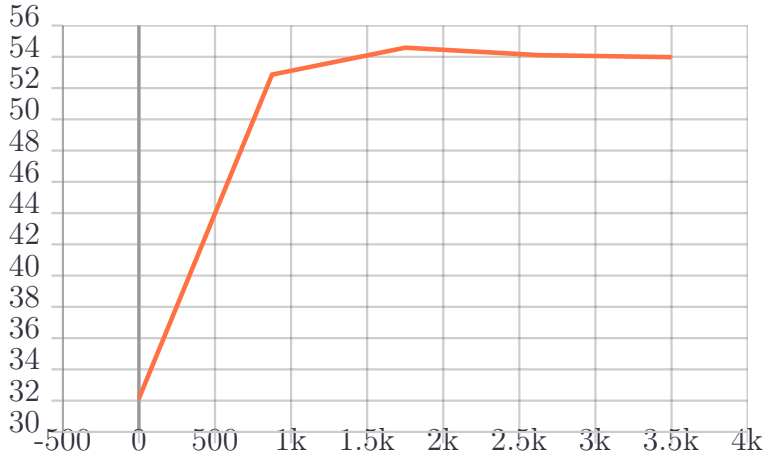
Figura 6.11: Risultati valutazione ROUGE con RelU con m pari a 32 su dataset COMMONGEN: (a) ROUGE1 (b) ROUGE2 (c) ROUGEL



(a)



(b)



(c)

Figura 6.12: Risultati valutazione ROUGE con ReLU m pari a 32 su dataset acalasia con TF-IDF: (a) ROUGE1 (b) ROUGE2 (c) ROUGEL

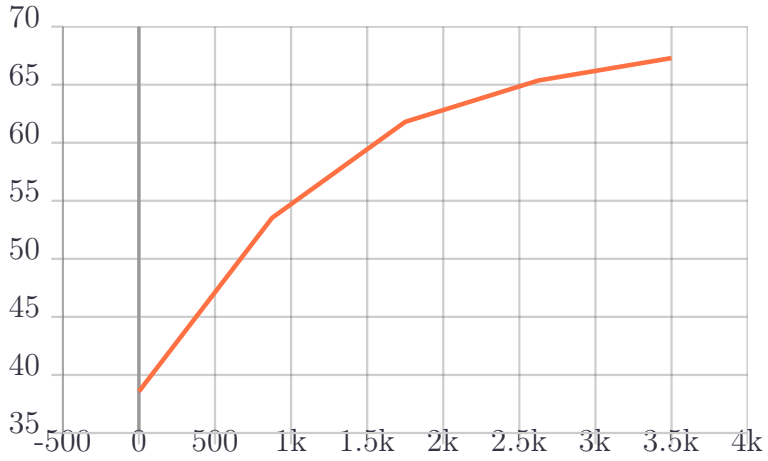
6.4.2 Il dataset con TF-IDF a livello di frase

Il secondo dataset comprende una selezione delle frasi basate su TF-IDF a livello di frase oltre che a livello di parola.

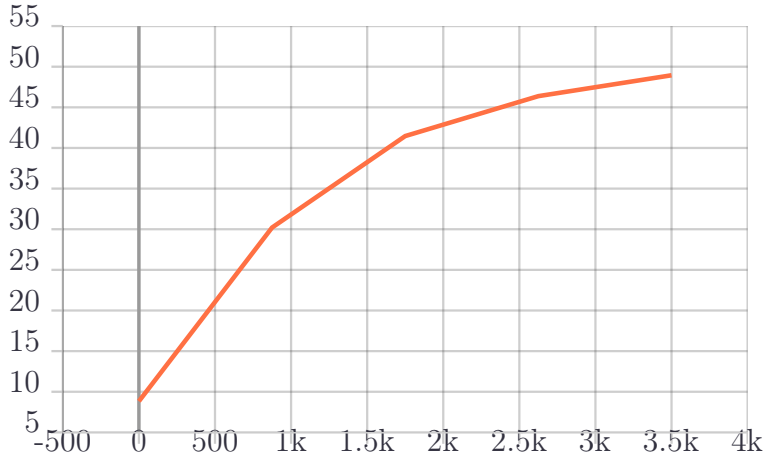
In questo caso, visti anche i risultati precedenti, si è deciso di provare solo il kernel ReLU con m pari a 64 e beam size pari a 4. Questi sono stati anche gli iperparametri scelti per poter confrontare gli addestramenti sui vari dataset. Con questo corpus si sono ottenuti i seguenti risultati con una ROUGE pari a 48.95, ROUGE2, e 60.07 per ROUGEL.

6.4.3 Il dataset con LSA

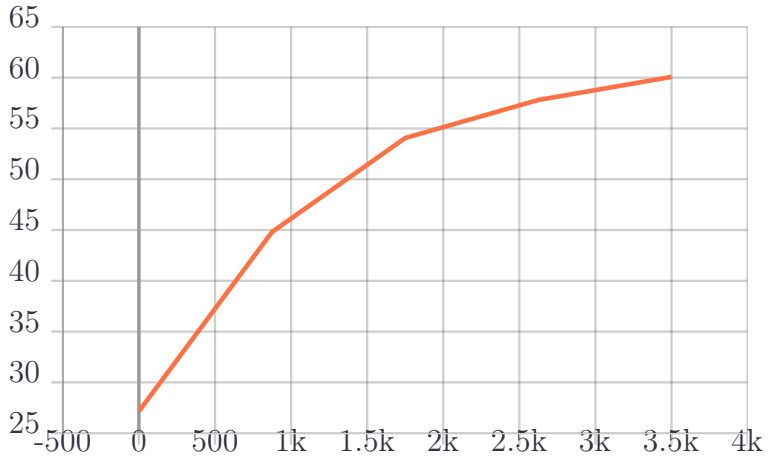
Infine si sono generati i set iniziali di parole utilizzando LSA. Con questa tecnica i risultati sono stati: 26.18, ROUGE2, e 42.92 su ROUGEL.



(a)

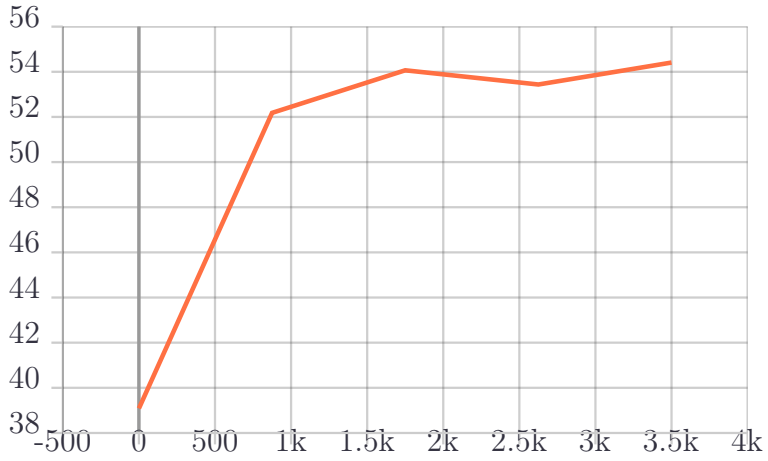


(b)

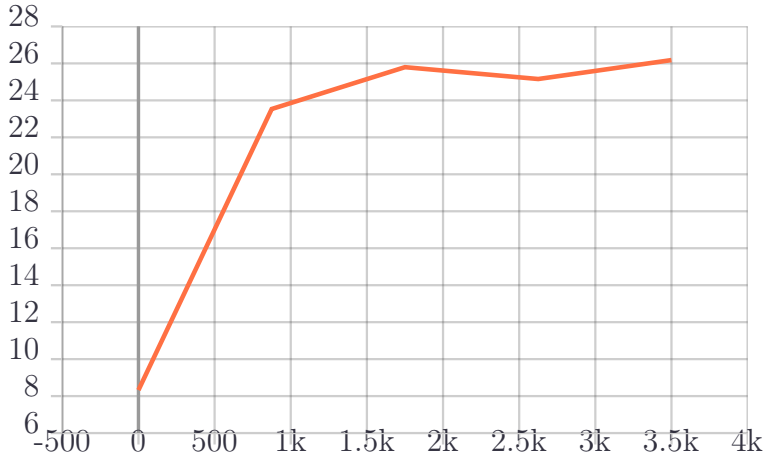


(c)

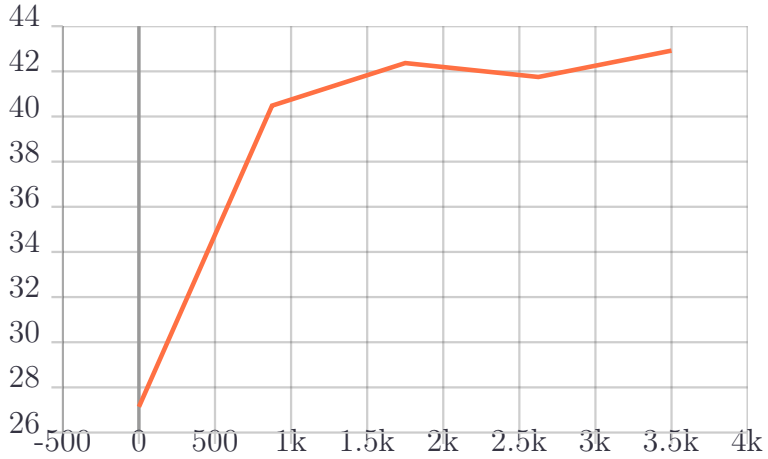
Figura 6.13: Risultati valutazione con ReLU con TF-IDF a livello di frase: (a) ROUGE1 (b) ROUGE2 (c) ROUGEL



(a)



(b)



(c)

Figura 6.14: Risultati valutazione con ReLu su LSA: (a) ROUGE1 (b) ROUGE2 (c) ROUGEL

6.5 La generazione di frasi

Terminati gli addestramenti sui tre dataset, sono state valutate le frasi generate dai vari modelli. Per fare questa operazione sono stati considerati come input i 138 gold standard che corrispondono all'output di POIROT e per ognuno di essi sono state generate 4 frasi diverse. Individuato l'output, si è effettuata una analisi manuale delle frasi considerando le seguenti caratteristiche [6]. In particolare, per il primo dataset:

- *fluency*: quanto una frase è fluente, indipendentemente dal significato corretto;
- *adequacy*: se una frase ha un corretto significato, indipendentemente dal fatto che sia fluente o meno;
- *interestingness*: valutazione di un eventuale contenuto di interesse per il dominio medico.

Per i dataset TF-IDF a livello di frase e LSA:

- *quality*: misurazione delle qualità complessiva della frase in termini di correttezza grammaticale e naturalezza;
- *repetitive*: quanto le parole all'interno della frase si ripetono o se rappresenta un pattern ripetuto;
- *consistency*: quanto la frase corrisponde al significato del set di input in termini di senso comune e logico;
- *factuality*: se la frase riflette o meno i fatti descritti nel contesto medico considerato (assenza di informazioni non corrette);
- *interestingness*: se la frase possiede o meno un contesto di interesse, indipendentemente dalle altre misure.

Per ognuna di queste misure è stato dato un valore binari, 0 o 1. Questa scelta è stata fatta per mantenere la valutazione il più semplice e oggettiva possibile, non introducendo sfumature di difficile interpretazione. La valutazione è stata svolta dal Dottor Giacomo Frisoni, considerato esperto in materia poichè membro del consiglio direttivo di AMAE, associazione per i malati di Acalsia Esofagea in Italia, e presidente dell'osservatorio per la qualità di vita e la consapevolezza digitale.

Il modello addestrato con TF-IDF

Il primo modello considerato, addestrato con set di input selezionati considerando la metrica TF-IDF, non ha riportato grossi risultati. Infatti, come visibile in Figura 6.15, le percentuali di frasi che hanno le caratteristiche considerate, riportate in Tabella 6.5 , sono di molto inferiore al 50%.

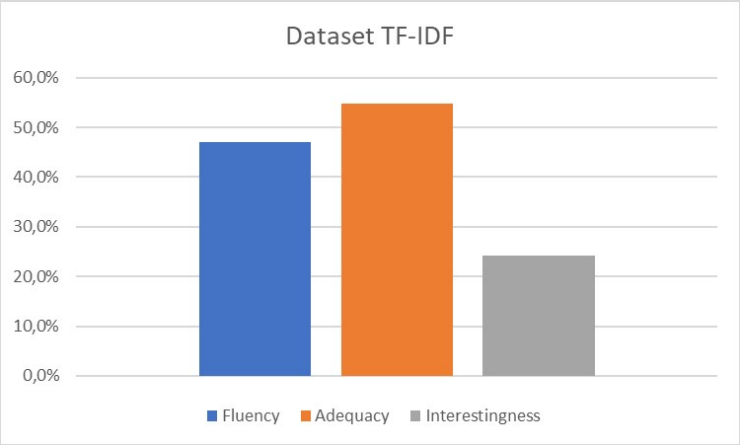


Figura 6.15: Media della valutazione sul dataset TF-IDF

Fluency	Adequacy	Interestingness
47.0%	54.8%	24.3%

Inoltre si è notato che nelle frasi considerate sono riportate forzatamente le frasi che compongono il set di input, senza alcuna coniugazione o variazione per migliorare il significato complessivo della frase.

Il modello addestrato con TF-IDF a livello di frase

Il secondo modello considerato, con dataset di addestramento basato su TF-IDF a livello di frase, il livello complessivo delle frasi è migliorato notevolmente. Come osservabile in Figura 6.16, le percentuali di frasi di output che possiedono una determinata caratteristica aumentano raggiungendo il 60%, i valori dettagliati cono riportati in Tabella 6.5. In particolare si può verificare che non vi è, in media, una netta distinzione fra qualità della prima frase predetta rispetto alle altre 3: nessun indice si discosta di molto dagli altri e le caratteristiche considerate sono presenti in tutte e quattro le predizioni svolte.

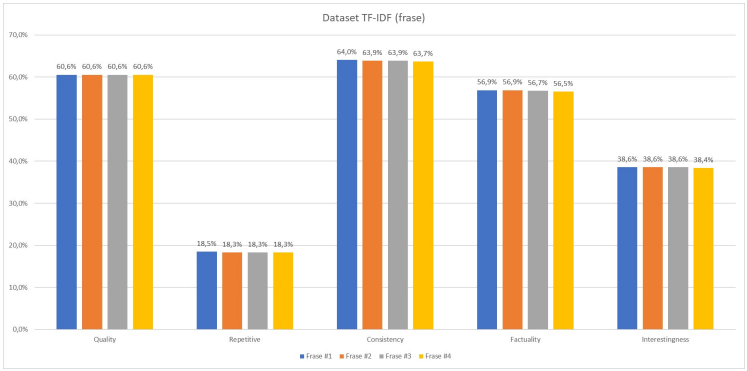


Figura 6.16: Media della valutazione sul dataset TF-IDF a livello di frasi

	Quality	Repetitive	Consistency	Factuality	Interestingness
1	60,6%	18,3%	63,7%	56,5%	38,4%
2	60,6%	18,3%	63,9%	56,7%	38,6%
3	60,6%	18,3%	63,9%	56,9%	38,6%
4	60,6%	18,5%	64,0%	56,9%	38,6%

Infine, analizzando le frasi prodotte dal modello, sono stati evidenziati i seguenti aspetti:

- sono presenti alcuni pattern ripetitivi che sono presenti nell'input e che sono stati appresi dalla rete ma che non sono sensati come: "emotelove", corrispondente a "<3" e quindi considerato come una emoticon, "a cl" come preposizione di luogo, e "short and long term differences"
- "25 to 60 years": si considera come una informazione molto interessante ma rappresenta un pattern ripetitivo e talvolta inserito fuori contesto;
- introducendo i numeri nei set di input, vengono gestite meglio alcune frasi e concetti importanti come l'esame della phmetria al quale viene associata la durata di 24 ore;
- si riscontra un comportamento migliore sulle frasi riguardanti l'assunzione di farmaci;
- vengono individuate associazioni corrette fra termini come "equipe" e "team" o "arm";
- la parole "Salvador" non viene associata al nome di un medico ma alla città brasiliana, probabilmente questo rappresenta un pattern introdotto dai pesi di partenza di COMMONGEN o una errata associazione del NER;

- il set di input "success rate" viene considerato correttamente a differenza del primo dataset.

I migliori risultati raggiunti sono stati riportati in Appendice C

Il modello addestrato con LSA

Infine sono state valutate le frasi di output del modello basato sul dataset costruito con LSA. Come visibile in Figura 6.17, le misurazioni effettuate riportano delle percentuali molto più alte rispetto al dataset precedente in ogni metrica, Tabella 6.5.

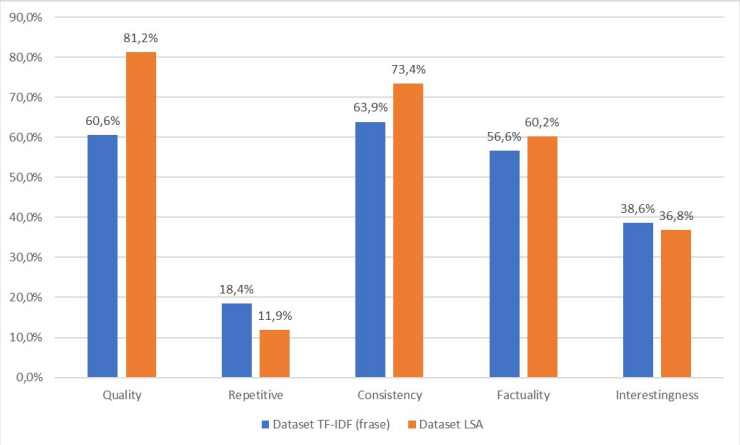


Figura 6.17: Media della valutazione sul dataset con LSA

	Quality	Repetitive	Consistency	Factuality	Interestingness
TF-IDF	60,6%	18,4%	63,9%	56,6%	38,6%
LSA	81,2%	11,9%	73,4%	60,2%	36,8%

Inoltre, anche in questo caso, Figura 6.18, non presenti differenze notevoli nei valori delle medie fra le 4 predizioni fatte per ogni input, Tabella 6.5.

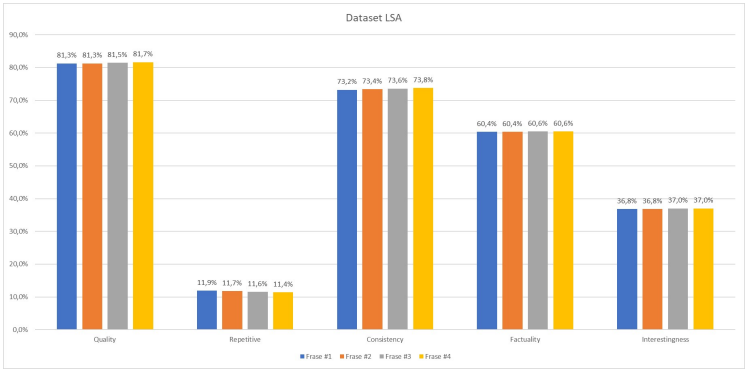


Figura 6.18: Media della valutazione sul dataset con LSA per indice di frase

	Quality	Repetitive	Consistency	Factuality	Interestingness
1	81,7%	11,4%	73,8%	60,6%	37,0%
2	81,5%	11,6%	73,6%	60,6%	37,0%
3	81,3%	11,7%	73,4%	60,4%	36,8%
4	81,3%	11,9%	73,2%	60,4%	36,8%

Con questo dataset, infine, si riportano le seguenti considerazioni:

- questo modello presenta il più alto livello di correttezza grammatica nelle frasi generate con una presenza minore di errori grammaticali o ripetizioni;
- il nome di professionisti medici vengono gestiti meglio e il NER viene riportato correttamente molto spesso;
- si osserva una maggiore aderenza alle parole indicate rispetto al TF-IDF e ad un migliore gestione del loro significato;
- si gestiscono in maniera ottimale alcune definizioni mediche e quindi si riescono a produrre delle migliori descrizioni di fatti in ambito medico come per i termini “peom”, “heller dor”, “fundoplication” e “LHM”;
- gli acronimi e le loro espansioni vengono associate al giusto significato, come nel caso di “ppi”.

Gli output più rilevanti sono stati riportati in Appendice D. Si può affermare, dunque, che con il dataset basato su LSA si è realizzato il conditional language model più stabile e maturo, in grado di generare spiegazioni di fatti medici nel modo migliore dai golden standard di POIROT.

Conclusioni e sviluppi futuri

I due principali contributi presentati in questa tesi sono: la verifica dei vantaggi prestazionali derivanti da Performer e la generazione di spiegazioni da set di parole correlate in ambito medico.

La loro realizzazione è stata possibile grazie ai seguenti passi. A seguito di un'analisi di FLAX, il recente framework proposto da Google per la realizzazione di reti neurali, si sono studiate le implementazioni dei modelli BERT e T5 basate su di esso (i.e., BERTX e T5X), oltre che il meccanismo di Attention FAVOR+. Dopo aver apportato miglioramenti al codice atti a estenderne le possibilità di utilizzo, si è introdotto il supporto a un'Attention modulare (lineare kernel Softmax, lineare kernel ReLU, o quadratica). Monitorando il consumo delle risorse in relazione alla lunghezza dell'input, è stato possibile osservare, anche in ambienti con ridotte capacità computazionali, una riduzione dei tempi di addestramento e di predizione, nonché una minore occupazione di memoria complessiva

Per l'addestramento dei vari modelli T5X considerati nel corso di questa ricerca, ci si è focalizzati su un caricamento pesi iniziale da altri modelli T5 pubblici (PyTorch, Attention quadratica) aventi già eseguito un fine-tuning su CommonGen. Ciò ha richiesto la definizione di una funzione di mappatura ispirata all'implementazione ufficiale HuggingFace che tenesse conto delle varianti architetturali dei modelli PyTorch e Flax. Dopo aver verificato che i risultati raggiunti con predizioni di test fossero gli stessi riportati in letteratura (ove possibile un confronto diretto) o analoghi in ordine di grandezza (ove non), sia con l'Attention ReLU che con l'Attention quadratica, ci si è concentrati sulla risoluzione del task considerato. In seguito all'introduzione di un meccanismo di multi-output nel modello e l'implementazione di una rouge apposita per valutare più frasi predette da un singolo concept set, si è osservato che il risultato migliore, sia in termini di valutazione a seguito del solo caricamento pesi (dove si sono replicati e superati i punteggi di ROUGE-F1 presentati sul paper originario di CommonGen) che in termini di risorse occupate con epoche aggiuntive sul medesimo dataset, è stato ottenuto con la funzione di Attention lineare ReLU e m, numero di proiezioni random da considerare, pari a 64. Determinato il kernel da utilizzare, si è ripetuto l'addestramento su tre dataset

“concept_set →” frasi" costruiti in modo non supervisionato a partire dai post e commenti pubblicati dai pazienti acalasici e dai loro caregiver. Durante tale fase si sono presi in considerazione sia approcci basati su TF-IDF che su norma LSA. La qualità dei tre conditional language model così ottenuti è stata valutata manualmente da esperti di dominio che hanno provveduto a esprimere un loro giudizio su 5 dimensioni diverse, tese a catturare sia abilità sintattiche che semantiche e di contenuto. Tale fase di test ha riguardato l'espansione a concept set dei gold standard usati per POIROT e l'assegnazione di un punteggio binario per 4 frasi generate da ciascuno di essi (beam_size = 4). Dagli addestramenti effettuati si è potuto osservare che il modello adestrato su dataset LSA ha raggiunto il miglior risultato riuscendo a predire frasi corrette dal punto di vista grammaticale e che sapessero descrivere e interpretare al meglio i set di parole di input. L'output è compost da frasi nelle quali è presente una conoscenza appresa dalla rete che permette di spiegare e esplicitare le relazioni semantiche presenti fra gli input ma aggiungendo anche informazioni aggiuntive che ampliano il senso e permetto al lettore di apprendere diverse informazioni utili.

In particolare nel task considerato e applicato al contesto medico dei gold standard di POIROT, si è osservata una tendenza delle rete neurale ad aggiungere una conoscenza nuova alle frasi, non limitandosi solamente a collegare le parole con delle semplici dipendenze, ma a “spiegarne” il contesto e coniugando molto spesso i termini o associando delle nozioni nuove non presenti nel dataset di partenza. Inoltre il modello, partendo da un set di parole di input, è in grado di spiegarle, comprendere il perché siano considerate correlate e produrre come output delle frasi corrette dal punto di vista grammaticale, di senso computo e, soprattutto, con informazioni interessanti e veritiere.

Con questo risultato si aprono diverse sviluppi futuri. Innanzitutto si possono valutare i risultati raggiunti facendoli valutare da una platea di esperti e pazienti più estesa e si può effettuare una analisi con metriche apprese o fisse più evolute di ROUGE, come per esempio METEOR. Inoltre, si può migliorare il risultato effettuando un ulteriore addestramento con un nuovo dataset. Per formare quest'ultimo si applica il concetto di densità kernel sullo spazio LSA ottenuto con POIROT, riconoscere in modo automatico delle n-uple di termini tra loro correlati, dunque estendendo il modello POIROT che lavora solo con coppie di termini in maniera incrementale. In questo modello, ogni termine viene considerato come una gaussiana e il grado di correlazione tra termini è l'intersezione fra le curve in uno spazio multidimensionale. Una volta identificati questi cluster, si ricercano le frasi che li contengono e si realizza il nuovo dataset. Questo approccio dovrebbe migliorare il risultato finale poichè si basa su un corpus che riflette al meglio l'operazione richiesta alla rete: l'input non è formato da termini estratti da frasi che hanno un alto potere informativo

ma, al contrario, si definiscono termini altamente correlati, come i gold standard di POIROT e si ricercano le frasi che li contengono.

Un altro ambito futuro di applicazione riguarda la produzione di dataset sintetici per event-extraction. In generale, gli eventi sono elementi simili alle relazioni ma più complessi: un evento corrisponde ad una entità che scatena con una azione altri eventi che a loro volta possono innescare della azioni. Il risultato è un ipergrafo con una struttura annidata. Nel campo medico un esempio può essere dato dal termine "farmaco" che innesta una reazione con un "effetto collaterale" che ha sua volta ha degli effetti negativi. Il lavoro svolto potrebbe essere applicato ad una rete che produce questo tipo di elementi così da generare frasi da rappresentazioni più complesse delle semplici relazioni e ampliare i dataset presenti in rete.

Infine si può utilizzare il lavoro svolto per la generazione di frasi che possano riassumere le informazioni contenute in una quantità elevata di documenti testuali. Per esempio, applicandolo al caso delle recensioni Amazon, si posso realizzare delle descrizioni sintetiche che riassumano tutte le caratteristiche principali, negative e positive, di un prodotto che vengono estratte dal parare di centinaia di utenti su un determinato prodotto. Con questa applicazione, si possono riconoscere gli elementi interessanti in un determinato contesto e in automatico generare una sintesi per ognuno di questi con un testo finale, eventualmente strutturato in una forma decisa e precisa (il 70% dei clienti ha questa opinione il 30% riporta questa affermazione e così via).

In conclusione, dato un insieme di documenti testuali non etichettati su un determinato dominio è possibile estrarre con POIROT, termini interessanti e gruppi di elementi fortemente correlati statisticamente tra di loro. Così facendo si possono riconoscere dei fenomeni di interesse e generare dei documenti intesi come sequenze/insiemi di termini caratterizzanti di quel fenomeno che si posso interpretare come delle spiegazioni per il fenomeno. Con il lavoro svolto in questa tesi, i documenti di output non sono insiemi di termini che hanno necessitano di un esperto di dominio per essere interpretati ma sono mappati in una frase sensata e di senso computo che sia in grado di spiegare l'argomento di interesse.

Appendice A

Il caricamento dei pesi

In questa appendice si riportano le corrispondenze individuate fra i layer delle implementazioni HuggingFace id BERT e T5 e quelle FLAX.

A.1 BERT

FLAX	HUGGINGFACE
embed	embeddings.word_embeddings
AddLearnPosiitonalEncodings_n	position_embedding-weight
LayerNorm_0 (x numero layer)	output.LayerNorm
MultiHeadDotProductAttention_1.key	attention.self.key
MultiHeadDotProductAttention_1.value	attention.self.value
MultiHeadDotProductAttention_1.query	attention.self.query
MultiHeadDotProductAttention_1.out	attention.self.output
LayerNorm_2	output.LayerNorm
MlpBlock_3.Dense_0	intermediate.dense.weight
MlpBlock_3.Dense_1	output.dense.weight
LayerNorm_n	ìoutput.LayerNorm
Dense_1	cls.predictions.transform.dense
LayerNorm_2	cls.predictions.LayerNorm
Dense_3	cls.predictions.decoder

In BERT per ogni livello indicato nella tabella è presente un array per i persi, *weights*, e uno per il bias, *bias*.

A.2 T5

FLAX	HUGGINGFACE
ENCODER_RELATIVE_POSEMB	EMBED_TOKEN
ENCODER_BLOCK_n (x numero layer)	EMBEDTOKENS
LAYER_NORM_0	T5BLOCK
SELF_ATTENTION_0	SELFATTENTION
QUERY['KERNEL']	Q
KEY['KERNEL']	K
VALUE['KERNEL']	V
OUT['KERNEL']	O
LAYERNORM_1	LAYER_NORM
MPL_0	DENSERELUDENSE
WI	WI
WO	WO
ENCODER_NORM	FINALNORM
DECODER_RELATIVE_POSEMB	EMBED_TOKEN
ENCODER_DECODER_n (x numero layer)	T5BLOCK
LAYER_NORM_0	LAYER_NORM
SELF_ATTENTION_0	SELF_ATTENTION
QUERY['KERNEL']	Q
KEY['KERNEL']	K
VALUE['KERNEL']	V
OUT['KERNEL']	O
LAYERNORM_1	LAYER_NORM
MULTIHEADDOTPRODUCTATTENTION_0	ENCDECATTENTION
QUERY['KERNEL']	Q
KEY['KERNEL']	K
VALUE['KERNEL']	V
OUT['KERNEL']	O
LAYERNORM_2	LAYER_NORM
MPL_0	DENSERELUDENSE
WI	WI
WO	WO
ENCODER_DECODER_NORM	FINALE_LAYER_NORM

Appendice B

I DockerFile per l'implementazione sul server

Di seguito si riporta i DockerFile utilizzati per portare il codice sul server e eseguire gli addestramenti.

B.1 BERTX

```
FROM python:3.7
LABEL maintainer="DISI NLU Research Group"

ENV DEBIAN_FRONTEND=noninteractive

RUN apt-get update -y && \
    apt-get install -y curl \
                       git \
                       bash \
                       subversion && \
    apt-get autoremove -y && \
    apt-get clean -y && \
    rm -rf /var/lib/apt/lists/*

RUN pip install numpy \
                torch==1.2.0+cu92 \
                torchvision==0.4.0+cu92 -f
                    https://download.pytorch.org/whl/torch_stable.html \
                flax \
                --upgrade git+https://github.com/google/flax.git \
                transformers \
```

```
              datasets \
              tqdm \
              jaxlib \
              tensorflow \
              tensorflow-datasets \
              matplotlib

RUN svn export
    https://github.com/google-research/google-research/trunk/protein_lm

ENV DEBIAN_FRONTEND=dialog

WORKDIR /home/ventani/flax-bert/build
```

B.2 T5X

```
FROM nvcr.io/nvidia/pytorch:20.09-py3
LABEL maintainer="DISI NLU Research Group"

ENV DEBIAN_FRONTEND=noninteractive

RUN apt-get update -y && \
    apt-get install -y curl \
                       git \
                       bash \
                       nano \
                       subversion && \
    apt-get autoremove -y && \
    apt-get clean -y && \
    rm -rf /var/lib/apt/lists/*

RUN pip install --upgrade pip
RUN pip install wrapt --upgrade --ignore-installed

RUN pip install numpy \
              #torch==1.7.1+cu110 \
              #torchvision==0.4.0+cu92 -f
                  https://download.pytorch.org/whl/torch_stable.html \
             flax \
              --upgrade git+https://github.com/google/flax.git \
              transformers \
              gin-config \
```

```
            dm-tree \
            datasets \
            tqdm \
            --upgrade jax jaxlib==0.1.57+cuda110 -f
                https://storage.googleapis.com/jax-releases/jax_releases.html
                \
            tensorflow==2.4.0 \
            tensorflow-datasets \
        tensorflow-text==2.4.0rc0 \
            matplotlib \
            ml-collections==0.1.0 \
            t5==0.7.1
ENV DEBIAN_FRONTEND=dialog
```

Appendice C

Le predizione del modello addestrato con dataset TF-IDF

Input: cardias esophagus muscle
Output: muscle hypertrophy of the esophagus and probable cardias involvement

Input: gastric mucosa esophagus
Output: mucous membrane hypertrophy of the esophagus and gastroesophageal reflux disease

Input: lower esophageal sphincter les
Output: esophageal achalasia esophagus dyskinesia of the distal lower sphincter and laparo surgery

Input: sternum chest bone
Output:
i have a strong pain in the sternum and chest and a little bit of pain in the bone
i have a strong pain in the sternum and chest and a bit of pain in the bone

Input: difficult swallowing dysphagia
Output: i have difficulty swallowing due to esophageal achalasia and esophagus dysphagia

Input: food regurgitation dysphagia
Output: regurgitation occurs due to retention of large amounts of saliva and dysphagia

Input: getting up meals regurgitate
Output: sometimes i regurgitate after meals i regurgitate getting up again

Input: achalasia rare disease esophagus neurodegenerative
Output:
this is from the wiki the most common form is primary esophageal achalasia which has no known underlying cause
a loved one is having symptoms similar to esophageal achalasia but in a completely different body region

Input: throat burning reflux
Output: i have a strong burning in my throat and gastroesophageal reflux disease

Input: gastroesophageal reflux sternum burning
Output:
i have a strong gastroesophageal reflux disease burning in the sternum and throat
burning in the sternum and throat due to gastroesophageal reflux disease

Input: acid reflux retrosternal burning
Output: do you have gastroesophageal reflux disease problems or retrosternal pain burning in the esophagus

Input: acid reflux heartburn
Output: i have gastroesophageal reflux disease and heartburn due to acid regurgitation

Input: achalasia esophagus nerve malfunction
Output:
esophageal achalasia esophagus dyskinesia of the vagus nerve
esophageal achalasia is usually found in patients 25 to 60 years of age but can be seen at any age with a nerve malfunction

Input: achalasia esophagus nerve destruction
Output:
esophageal achalasia characterized by minimal contractility along the esophagus body type ii nerve damage
a nerve damage in the esophagus is reported by a considerable number of esophageal achalasia patients

Input: achalasia sternum pain
Output: i often have pain in the sternum due to esophageal achalasia

Input: achalasia diagnosis gastroscopy
Output: the diagnosis of esophageal achalasia is done with a gastroscopy

Input: manometry diagnosis resolution achalasia
Output: i made the high resolution manometry that confirm the diagnosis of esophageal achalasia

Input: phmetria examination hours
Output: phmetria 24 hours after the examination
this morning i have the phmetria and a reflux examination and a manometry
this morning i have the phmetria to verify that the reflux valve is normal
this morning i have the phmetria for the reflux examination
this morning i have the phmetria and the reflux examination

Input: barium x-ray diagnosis achalasia
Output: diagnosis of esophageal achalasia with barium and x ray chest

Input: barium radiography diagnosis achalasia
Output: i made the diagnosis with barium radiography and it confirms that i have esophageal achalasia

Input: esomeprazole drug gastroesophageal reflux
Output: esomeprazole is a drug that prevents gastroesophageal reflux disease

Input: pantoprazole drug acid reflux treat
Output: they refer to the pantoprazole as a drug that helps in treating gastroesophageal reflux disease

Input: ranitidine gastroesophageal reflux
Output: ranitidine is a gastroprotector for those suffering from gastroesophageal reflux disease

Input: nifedipine drug relax muscle
Output: relax the muscle a little with the nifedipine drug

Input: tomato avoid reflux
Output: i eat everything to avoid reflux i avoid a bit the meaty foods such as

cream cheese yogurt etc

Input: avoid spicy food
Output: i avoid spicy food and pizza because they make me feel bad

Input: amae association achalasia
Output: the amae is the association that deals with esophageal achalasia

Input: achalasia disease rare
Output: esophageal achalasia is a rare disease and it is often found in patients 25 to 60 years of age

Input: achalasia chicago classification types
Output:
and i went to a gastroenterology in milan for the classification of esophageal achalasia and esophagus types
and i went to a gastroenterology in milan to do the classification of esophageal achalasia type ii
and i went to a gastroenterology in milan for the classification of esophageal achalasia and esophagus type ii
and i went to a gastroenterology in milan and they assigned me to the three types of esophageal achalasia

Input: achalasia embarassment meals
Output:
i have been suffering from esophageal achalasia for years and every meal is a bit of an embarassment for me
i have been suffering from esophageal achalasia for years and every meal is a bit of a embarassment
i have been suffering from esophageal achalasia for many years and every meal is a bit of an embarassment
i have been suffering from esophageal achalasia for years and every meal is a bit of an embarassment

Input: abano terme professor ancona
Output: but in january 2015 i had an operation in abano terme by the professor ancona

Input: abano terme professor zaninotto
Output: but in january 2015 i had an operation in abano terme by the professor

zaninotto

Input: bologna professor capizzi
Output: and i highly recommend the team of professor capizzi of the institute of bologna

Input: professor costantini achalasia
Output: the professor costantini told me about esophageal achalasia

Input: professor zaninotto padua
Output: professor zaninotto is from padua

Input: padua equipe costantini
Output: i had an operation in padua with dr costantini and dr ancona and his team

Input: san donato milanese bonavina
Output:
i was operated on june 17 in san donato milanese milan by prof bonavina i ve started living again
and i highly recommend the team of professor bonavina of the institute of san donato milanese milan

Input: padua esophagus disease center
Output: in padua there is a center for esophagus disease

Input: costantini padua hospital
Output: i had an operation at the civil hospital in padua by doctor costantini

Input: reflux depressive disorder
Output: i suffer from gastroesophageal reflux disease and depressive disorder

Input: depression anxiety gastroesophageal reflux
Output: i suffer from anxiety and if i have gastroesophageal reflux disease i don t think it s just anxiety

Input: achalasia botulinum injection
Output: i did the manometry and the botulinum toxin injections for esophageal achalasia

Input: endoscopic botulinum injection
Output: i had an operation in september 2010 at humanitas i had an endoscopic injection of botulinum toxin

Input: repeated botulinum injections
Output: i have had repeated injections of botulinum toxin for many years

Input: myotomy achalasia surgery
Output: myotomy is a type of surgery that solves the problem of esophageal achalasia

Input: heller dor laparoscopy
Output:
i did the heller dor in video laparo surgery
i did the heller dor in video laparoscopy

Input: heller myotomy success rate
Output:
heller dor myotomy video laparo surgery with a high success rate short and long term differences with heller s myotomy video
laparo surgery with relative success rate

Input: heller dor recovery time
Output: for the first time after the operation heller dor the recovery time has come

Input: poem endoscopic myotomy
Output: poem endoscopic myotomy of the esophagus

Input: poem antireflux plastic
Output:
the poem does not make antireflux plastic and does not work
the poem does not make antireflux plastic

Input: poem long term results
Output:
the long term results of the poem were almost equivalent to those obtained in the short term
and long term post surgery assess the results of the poem

Input: poem long term results unknown
Output:
the long term results of the poem were unknown
the long term results of the poem are unknown
long term post surgery assess the degree of satisfaction and the favorable change in the quality of life
and long term post surgery assess the degree of satisfaction and the favorable change in the quality of life

Input: poem success rate
Output:
the rate of success of the poem is about 96

Input: poem recovery time
Output:
the recovery time for the poem was a short one

Appendice D

Le predizione del modello addestrato con dataset LSA

Input: gastric mucosa esophagus
Output: esophagus mucosa and gastric mucosa are two different things

Input: lower esophageal sphincter les
Output:
the lower part of the esophageal sphincter is often seen with laparotomy
the lower part of the esophageal sphincter has been found to have spasms

Input: difficult swallowing dysphagia
Output:
the dysphagia makes swallowing difficult especially when compared to before
the dysphagia makes swallowing difficult

Input: solid liquid dysphagia
Output: i have dysphagia both with solid and liquid

Input: food regurgitation dysphagia
Output: dysphagia and regurgitation of undigested food residues

Input: getting up meals regurgitate
Output:
i have been getting up since i was 12 to regurgitate during meals
i have been getting up since i was 12 to regurgitate after meals

Input: diverticulum obstacle food descent
Output: the obstacle that creates problems for food descent is the diverticulum

Input: stomach inflammation esophagitis
Output:
esophagitis is a form of inflammation that goes down the stomach mucous membrane
i have esophagitis and stomach inflammation

Input: loss esophagus motility
Output: loss of esophagus motility and consequent stiffening

Input: throat burning reflux
Output: i have burning in the throat due to reflux

Input: gastroesophageal reflux sternum burning
Output: burning in the sternum and gastroesophageal reflux disease

Input: acid reflux retrosternal burning
Output: i have retrosternal asthma attacks and burning and acid reflux retrosternal pain and acid reflux bronchitis and burning

Input: achalasia nerve inflammation
Output:
i think it is due to the inflammation of the vagus nerve and a consequence of achalasia
i think it is due to the inflammation of the nerve ending up in the achalasia

Input: achalasia esophagus nerve malfunction
Output: the malfunction of the nerve endings of the esophagus is typical of schalasia

Input: achalasia esophageal sphincter failure relax
Output:
the failure of the esophageal achalasia is due to the fact that the sphincter does not relax
the failure of the esophageal achalasia was due to the failure of the sphincter to relax

Input: achalasia sternum breathe
Output: and i have to breathe deeply because the sternum is always in turmoil due to achalasia

Input: achalasia sternum pain
Output:
i have always had pain in the sternum as if i had been in a battle with achalasia
the pain in the sternum is typical of achalasia

Input: gastritis burning feeling stomach
Output: the feeling of gastritis in the stomach and burning sensation

Input: esophageal candidiasis infection esophagus
Output: esophageal candidiasis is due to an infection of the esophagus mucous membrane

Input: achalasia diagnosis gastroscopy
Output: i did a gastroscopy to get the diagnosis of achalasia

Input: gastroscopy flexible tube procedure
Output:
gastroscopy is a fairly simple and relatively straight forward procedure with a tube that is very elastic
gastroscopy is a relatively simple and relatively straight forward procedure with a tube that is very elastic

Input: high resolution manometry
Output: the manometry must be done at high resolution

Input: manometry diagnosis resolution achalasia
Output: the high resolution manometry gives the diagnosis of achalasia

Input: nose tube manometry
Output:
manometry and phmetry with the tube in the nose
manometry and phmetry with tube in the nose

Input: phmetria examination hours
Output: i did the phmetria 24 hours after the examination and everything is fine

Input: barium radiography diagnosis achalasia
Output:
barium radiography gave me the diagnosis of achalasia
barium radiography confirmed the diagnosis of achalasia

Input: esomeprazole drug gastroesophageal reflux
Output:
esomeprazole is an effective drug against gastroesophageal reflux
esomeprazole is an effective drug against gastroesophageal reflux disease

Input: lansoprazole ppi drug acid
Output:
i am taking lansoprazole ppi which is an analgesic drug for oral acid
i am taking lansoprazole ppi which is a drug closely related to acid

Input: pantoprazole drug acid reflux treat
Output: pantoprazole is an effective drug to treat acid reflux

Input: proton pump inhibitor acid reflux
Output: the proton pump enzyme inhibitor is used to reduce acid reflux

Input: ranitidine gastroesophageal reflux
Output: ranitidine solves gastroesophageal reflux problems

Input: nifedipine chest pain
Output: i have been taking nifedipine for some pain in the chest for some years now

Input: alcohol consumption risk esophagitis
Output: the risk of esophagitis is increasing with the consumption of alcohol

Input: lemon acidity reflux
Output: i take a teaspoon of citrullus lanatus and a teaspoon of rabeprazole for acidity reflux

Input: rice throat blocked
Output: and i had my throat blocked with a teaspoon of rice s and i was blocked with rice in my throat

Input: wine stomach acid
Output:
and i have to be careful not to blend the acidity of the stomach with the white wine
and i have to be careful not to blend the acidity of the stomach with the sparkling wine

Input: drink glass water
Output: and i have to drink a glass of water to get everything down

Input: herbal teas relax muscles
Output:
i use erythroxylum coca cola drink and relax the muscles with herbal teas relaxing teas or herbal teas relax the muscles
relaxing herbal teas relax the muscles

Input: achalasia healthy nutrition
Output:
i wanted to know what nutrition and healthy lifestyle are for those with achalasia
i wanted to know if any of you have tried healthy nutrition for achalasia
i wanted to know what nutrition and healthy lifestyle are for those with achalasia like us
i wanted to know if any of you have been followed for nutrition and healthy achalasia

Input: achalasia diet reflux
Output: i have been on a diet for a year and a half and i have no problems with reflux disease

Input: amae association achalasia
Output: amae is the non profit association for the achalasia

Input: achalasia chicago classification types
Output: in chicago there are three types of achalasia according to the classification

Input: achalasia diagnosis time
Output: the diagnosis of achalasia was given to me in time

Input: share experiences achalasia group
Output:
i wanted to ask you if any of you have experiences to share with this group a group of achalasia experts who share experiences and knowledge

Input: abano terme professor ancona
Output: professor ancona in abano terme is very good

Input: brescia doctor fumagalli
Output: today i went to the doctor fumagalli in brescia for a consultation

Input: doctor fumagalli achalasia
Output: today i went to the doctor fumagalli who diagnosed me with achalasia

Input: giovanni sarnelli naples
Output: giovanni sarnelli at the ssc naples conference

Input: professor costantini padua
Output: professor mario costantini in padua

Input: rome doctor familiari
Output:
i would like to know if any of you have been followed by doctor familiari in rome
hello everyone i wanted to ask you if any of you have contacts with doctor familiari in rome

Input: rome doctor costamagna
Output: doctor costamagna in rome practicing for 20 years

Input: rome costamagna equipe
Output: equipe dr costamagna in rome

Input: rozzano doctor bonavina
Output: i went to doctor bonavina in rozzano and he gave me directions

Input: padua esophagus disease center
Output:
padua is the best center in the world for esophagus disease
in padua there is the best center for esophagus disease
padua is the best center for esophagus disease in europe

Input: rozzano humanitas hospital
Output: the humanitas hospital in rozzano is one of the best

Input: padua heller dor
Output: i did heller dor in 2005 in padua

Input: reflux depressive disorder
Output: i have depressive disorder and heavy reflux

Input: stress acid reflux Output: i am under stress because i have to be able to reduce my acid reflux and i have to be careful not to excess acid reflux due to stress biology

Input: achalasia botulinum injection
Output: i did botulinum toxin injection twice a year to see if it can help me with achalasia

Input: endoscopic botulinum injection
Output: endoscopic injection of botulinum toxin

Input: repeated botulinum injections
Output: i have had repeated injections of botulinum toxin

Input: myotomy achalasia surgery
Output: i am considering a possible myotomy surgery for achalasia in laparoscopy

Input: poem or heller dor
Output: i would like to know if you did the poem or the heller dor

Input: heller dor laparoscopy
Output: i did heller dor laparoscopy and everything went well

Input: heller dor laparoscopic myotomy
Output: i did heller dor myotomy in laparoscopy three years ago

Input: heller dor antireflux plastic
Output:
i did heller dor with antireflux plastic
i did heller dor with antireflux plastic in 2010

Input: heller dor fundoplication
Output:
i would like to know if any of you have done the fundoplication with heller dor
i would like to know if any of you have done the heller dor fundoplication

Input: heller myotomy dor fundoplication
Output: myotomy according to heller and fundoplication according to dor

Input: heller myotomy success rate
Output:
myotomy according to heller has a high success rate and safety
the success rate of heller myotomy was 80
the success rate of heller myotomy is 80

Input: heller myotomy recovery time
Output:
after heller myotomy what time does it take to recover
after heller myotomy what recovery time did you have

Input: heller dor recovery days
Output: i did heller dor two days ago and today i m fine good recovery

Input: poem endoscopic myotomy
Output: the poem is an endoscopic myotomy

Input: poem antireflux plastic
Output: the poem does not include antireflux plastic

Input: poem experimental techniques
Output:
i wanted to know if any of you have tried the new experimental techniques for poem
i wanted to know if any of you have tried the poem or any other experimental techniques
i wanted to know if any of you have tried the poem or experimental techniques

Input: poem long term results
Output:
the long term results of the poem should be reported as soon as possible
the long term results of the poem are not immediately known

Input: poem long term results unknown
Output:
the long term results of the poem are still unknown
unfortunately the long term results of the poem are unknown
the long term results of the poem are unknown

long term results of the poem are unknown

Input: poem recovery time
Output:
i did the poem some time ago and i m fine good recovery
good recovery time for those who have done the poem
good recovery time for those who made the poem

Input: achalasia gastroprotector life
Output: the gastroprotector is a life saver

Ringraziamenti

Il primo ringraziamento va al relatore di questo lavoro, il Prof. Gianluca Moro, che ha reso possibile tutto ciò e ha acceso il mio personale interesse nei confronti di questa splendida disciplina. Ringrazio inoltre il Dottor Giacomo Frisoni che ha lavorato con me in questo progetto negli ultimi 5 mesi: mi ha sempre spinta a raggiungere risultati migliori, a districarmi in questo mondo affascinante e complesso e a non mollare mai. Infine ringrazio la mia famiglia e i miei amici e colleghi dell'ufficio che mi hanno supportato in questo percorso, specialmente nell'ultimo anno e mezzo, e hanno sempre creduto in me.

Bibliografia

[1] Sara Crouse et al. *Common Crawl*. (accessed: 02.21.2021).

[2] Jay Alammar. *The Illustrated Transformer*. (accessed: 03.17.2021).

[3] Xavier Bourret Sicotte et al. Alexey Grigorev. *How to intuitively explain what a kernel is?* (accessed: 02.21.2021).

[4] Iz Beltagy, Matthew E Peters e Arman Cohan. "Longformer: The long-document transformer". In: *arXiv preprint arXiv:2004.05150* (2020).

[5] Miguel Romero Calvo. *Dissecting BERT Part 1: The Encoder*. (accessed: 02.21.2021).

[6] Asli Celikyilmaz, Elizabeth Clark e Jianfeng Gao. "Evaluation of text generation: A survey". In: *arXiv preprint arXiv:2006.14799* (2020).

[7] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. A cura di Alessandro Moschitti, Bo Pang e Walter Daelemans. ACL, 2014, pp. 1724–1734. DOI: 10.3115/v1/d14-1179. URL: https://doi.org/10.3115/v1/d14-1179.

[8] Krzysztof Choromanski et al. "Rethinking attention with performers". In: *arXiv preprint arXiv:2009.14794* (2020).

[9] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[10] Giacomo Frisoni, Gianluca Moro e Antonella Carbonaro. "Learning Interpretable and Statistically Significant Knowledge from Unlabeled Corpora of Social Text Messages: A Novel Methodology of Descriptive Text Mining". In: ().

[11] Giacomo Frisoni, Gianluca Moro e Antonella Carbonaro. "Towards Rare Disease Knowledge Graph Learning from Social Posts of Patients". In: *Research and Innovation Forum 2020: Disruptive Technologies in Times of Change*. Springer International Publishing. 2021, pp. 577–589.

[12] Giacomo Frisoni, Gianluca Moro e Antonella Carbonaro. "Unsupervised Descriptive Text Mining for Knowledge Graph Learning". In: (2020).

[13] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* $_{C}$ *oncepts*, $Tools$, $and Techniques to Build Intelligent Systems$. O'Reilly Media, 2019.

[14] Sepp Hochreiter e Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[15] Nikita Kitaev, Łukasz Kaiser e Anselm Levskaya. "Reformer: The efficient transformer". In: *arXiv preprint arXiv:2001.04451* (2020).

[16] Simeon Kostadinov. *Inderstanding Encoder Decoder seq2seq model*. (accessed: 02.21.2021).

[17] Bill Yuchen Lin et al. "Commongen: A constrained text generation challenge for generative commonsense reasoning". In: *arXiv preprint arXiv:1911.03705* (2019).

[18] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of summaries". In: gen. 2004, p. 10.

[19] Ali Madani et al. "ProGen: Language Modeling for Protein Generation". In: *bioRxiv* (2020). DOI: 10.1101/2020.03.07.982272. eprint: https://www.biorxiv.org/content/early/2020/03/13/2020.03.07.982272.full.pdf. URL: https://www.biorxiv.org/content/early/2020/03/13/2020.03.07.982272.

[20] Matthew E Peters et al. "Deep contextualized word representations". In: *arXiv preprint arXiv:1802.05365* (2018).

[21] *Pretrained models*. (accessed: 02.21.2021).

[22] Ratish Puduppully, Li Dong e Mirella Lapata. "Data-to-text Generation with Entity Modeling". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, lug. 2019, pp. 2023–2035. DOI: 10.18653/v1/P19-1195. URL: https://www.aclweb.org/anthology/P19-1195.

[23] Alec Radford et al. "Improving language understanding by generative pre-training". In: (2018).

[24] Colin Raffel et al. “Exploring the limits of transfer learning with a unified text-to-text transformer”. In: *arXiv preprint arXiv:1910.10683* (2019).

[25] Google Research. *BERT*. URL: https://github.com/google-research/bert. (accessed: 02.21.2021).

[26] The Flax authors Revision. *Flax documentation*. (accessed: 02.24.2021).

[27] Alexander Rives et al. “Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences”. In: *bioRxiv* (2020). DOI: 10.1101/622803. eprint: https://www.biorxiv.org/content/early/2020/08/31/622803.full.pdf. URL: https://www.biorxiv.org/content/early/2020/08/31/622803.

[28] Ilya Sutskever, Oriol Vinyals e Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. A cura di Zoubin Ghahramani et al. 2014, pp. 3104–3112. URL: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.

[29] Fardin Syed, Riccardo Di Sipio e Pekka Sinervo. “Bidirectional Long Short-Term Memory (BLSTM) neural networks for reconstruction of top-quark pair decay kinematics”. In: *arXiv preprint arXiv:1909.01144* (2019).

[30] Yi Tay et al. “Efficient transformers: A survey”. In: *arXiv preprint arXiv:2009.06732* (2020).

[31] Programmatori Transformer. *Transformer*. (accessed: 02.24.2021).

[32] Barak Turovsky. *Found in translation: More accurate, fluent sentences in Google Translate*. URL: https://blog.google/products/translate/found-translation-more-accurate-fluent-sentences-google-translate/. (accessed: 01.03.2021).

[33] Kiyotaka Uchimoto, Satoshi Sekine e Hitoshi Isahara. “Text generation from keywords”. In: *COLING 2002: The 19th International Conference on Computational Linguistics*. 2002.

[34] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. A cura di Isabelle Guyon et al. 2017, pp. 5998–6008. URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need.

[35] Sinong Wang et al. “Linformer: Self-attention with linear complexity”. In: *arXiv preprint arXiv:2006.04768* (2020).

[36] Sam Wiseman e Alexander M. Rush. “Sequence-to-Sequence Learning as Beam-Search Optimization”. In: *CoRR* abs/1606.02960 (2016). arXiv: 1606.02960. URL: http://arxiv.org/abs/1606.02960.

[37] W. Xiang e B. Wang. “A Survey of Event Extraction From Text”. In: *IEEE Access* 7 (2019), pp. 173111–173137. DOI: 10.1109/ACCESS.2019.2956831.

[38] Manzil Zaheer et al. “Big bird: Transformers for longer sequences”. In: *arXiv preprint arXiv:2007.14062* (2020).

[39] Grace Zhang. *What is the kernel trick? Why is it important?* (accessed: 02.21.2021).