ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

SCILAY: UN NUOVO DATASET PER LONG DOCUMENT SUMMARIZATION SCIENTIFICI E DIVULGATIVI DI STUDI BIOMEDICI

Elaborato in
Programmazione Di Applicazioni Data Intensive

Relatore
Prof. Gianluca Moro

Co-relatore
Dott. Paolo Italiani, Luca Ragazzi e Giacomo Frisoni

Presentata da
Mattia Panni

Terza Sessione di Laurea
Anno Accademico 2022 – 2023

PAROLE CHIAVE

Abstractive Summarization

Natural Language Processing

Deep Neural Networks

Seq2Seq Models

Large Language Models

Il problema non è l'ascesa delle macchine "intelligenti",
ma l'instupidimento dell'umanità.
Astra Taylor[1].

Abstract

Questa tesi esplora il prospero settore dell'intelligenza artificiale (AI) e dell'elaborazione del linguaggio naturale (NLP), con l'obiettivo di automatizzare la creazione di riassunti comprensibili per gli articoli scientifici. Sfruttando metodologie AI all'avanguardia, questo studio si propone di ridurre il divario tra i dettagli intricati racchiusi nelle pubblicazioni scientifiche e la loro comprensione da parte del grande pubblico. La complessità e la profondità tecnica degli articoli accademici spesso costituiscono una barriera alla comprensione generalizzata. Mentre gli abstract tecnici si rivolgono bene agli esperti del settore, possono risultare criptici per i "non addetti ai lavori". Affrontando questo divario comunicativo, la tesi propone un nuovo dataset denominato *SciLay* il cui scopo è quello di fornire una base solida per l'addestramento di modelli di generazione automatica di riassunti laici (o lay) e tecnici di ricerche scientifiche.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Di seguito, si delineano le basi concettuali e le motivazioni che hanno ispirato questa tesi, fornendo al lettore una panoramica del percorso di ricerca e delle innovazioni tecnologiche nell'ambito dell'intelligenza artificiale e del natural language processing, che hanno reso possibile l'ambizioso obiettivo di automatizzare la sintesi di riassunti accessibili di articoli scientifici.

1.1 Intelligenza Artificiale

L'*intelligenza artificiale* (AI) è un campo interdisciplinare che si concentra sulla creazione di sistemi o macchine in grado di eseguire attività che richiedono, fra le altre cose

- **Apprendimento automatico**. Le AI utilizzano tecniche di apprendimento automatico per acquisire conoscenze dai dati. Questo significa che anziché essere programmate esplicitamente per eseguire compiti specifici, come accade per gli algoritmi tradizionali, le AI apprendono dai dati e migliorano le loro prestazioni con l'esperienza.
- **Generalizzazione**. Le AI sono in grado di adattarsi a nuove situazioni e compiti. Possono generalizzare le loro conoscenze per risolvere problemi simili a quelli affrontati durante l'addestramento.
- **Processamento dei dati non strutturati**: Le AI sono spesso utilizzate per analizzare e comprendere dati non strutturati, come testo, immagini e suoni. Possono estrarre informazioni significative da fonti di dati complesse e rumorose.

Queste proprietà, auspicabili per tali sistemi, sono comunemente riconosciute come indicatori di intelligenza, da cui deriva il termine "Intelligenza Artificiale".

Questo campo è stato oggetto di ricerca e sviluppo fin dagli anni '50 ed è cresciuto enormemente in importanza e complessità. L'AI ha rivoluzionato molti aspetti della nostra vita, dai motori di ricerca online alla guida autonoma, dalla diagnosi medica all'automazione industriale.

1.1.1 Storia della nascita delle AI

L'idea di creare macchine che possano simulare l'intelligenza umana ha radici antiche, ma il campo dell'AI moderna è iniziato nel 1956 con una conferenza tenuta a Dartmouth College, dove i ricercatori hanno discusso di "programmare un computer per simulare l'intelligenza umana". Questo evento è spesso considerato l'inizio ufficiale dell'AI come campo di studio.

Negli anni '50 e '60, gli studiosi dell'AI erano ottimisti riguardo alle potenzialità della disciplina. Credevano che sarebbe stato relativamente semplice programmare un computer per eseguire compiti che richiedevano intelligenza, come giocare a scacchi o tradurre lingue. Tuttavia, presto divenne chiaro che molte delle attività che gli esseri umani eseguono con facilità sono estremamente complesse da programmare.

Questo ha portato a un periodo noto come "inverno dell'AI" negli anni '70 e '80, durante il quale il finanziamento e l'interesse per l'AI diminuirono notevolmente. Ma negli anni '90, con l'aumento della potenza di calcolo, lo sviluppo di nuove tecniche e la disponibilità sempre maggiore di dati, l'AI ha conosciuto una rinascita. Oggi, l'AI è uno dei campi di ricerca e sviluppo più attivi al mondo.

1.1.2 Limiti e Potenzialità delle AI

D'altrocanto, come accade per ogni cosa, questo tipo di sistemi ottengono grandi risultati per un certo tipo di problemi, ma non per altri. Quando l'ambiente è stabile, ovvero presenta regole e/o pattern fissi (come nel caso degli scacchi), le AI eguagliano se non addirittura battono le performance degli esseri umani. Tuttavia, in situazioni colme di incertezza, potenza di calcolo elevata e disponibilità di grandi moli di dati contribuiscono in maniera limitata. In altre parole, al verificarsi di "Cigni neri"[2], ovvero situazioni altamente improbabili ma altrettanto impattanti delle quali, per loro natura, non disponiamo dati, le AI sono "fragili", ossia vengono danneggiate da tali eventi. Il termine "Cigno nero" è stato coniato per mettere in evidenza la nostra tendenza a sottovalutare eventi rari ma estremamente influenti e il nostro desiderio retrospettivo di spiegare tali eventi come se fossero stati prevedibili in anticipo. Abbiamo piuttosto "[...] bisogno di uno sguardo e di un coraggio che ci permettano di rimanere intelligenti in un mondo intelligente"[1], da qui la dedica di questo

lavoro. "Non è tempo di metterci comodi e rilassarci[...]"[1], occorre piuttosto comprendere e gestire il rischio associato a tali eventi, piuttosto che presumere che le AI possano prevederli o gestirli autonomamente.

Negli ultimi tempi, il *Machine Learning* (ML) è emerso come una macro-area tecnica di primaria importanza nell'ambito dell'AI. Le sue applicazioni di successo spaziano in diversi settori, tra cui il riconoscimento vocale e la visione artificiale.

Tuttavia, l'efficacia del machine learning può risultare limitata quando si affrontano dati complessi come immagini o testi in linguaggio naturale. La preparazione dei dati e la selezione delle variabili rilevanti richiedono spesso l'intervento di esperti e un notevole sforzo. In queste circostanze, il *Deep Learning* (DP) si è rivelato una risorsa preziosa. Questo approccio è in grado di estrarre autonomamente le variabili più importanti direttamente dai dati grezzi, aprendo nuove prospettive di ricerca.

Il deep learning è un campo relativamente giovane, con potenzialità ancora inesplorate, in grado di eseguire analisi dei dati a un livello più profondo. L'applicazione del deep learning ha portato a miglioramenti significativi in numerosi settori. Ha reso possibile lo sviluppo di auto a guida autonoma, assistenti virtuali capaci di comprensione del linguaggio naturale e macchinari medici in grado di identificare masse tumorali con una precisione superiore a quella umana.

Entrambe le macro-aree sono figlie dell'enorme impulso derivato dall'analisi dei dati e dalla capacità di estrarre conoscenza da voluminose masse di informazioni. In questo contesto, la disponibilità di dati è diventata cruciale, e l'abbondanza di dati ha reso possibile l'estrazione di conoscenze estremamente precise.

Questo cambiamento è stato alimentato dalla crescente generazione di dati, che avanza con una velocità precedentemente inimmaginabile. Questo scenario ha portato alla necessità di sviluppare i nuovi metodi citati in precedenza, per gestire e analizzare dati le cui dimensioni superano ampiamente le capacità umane.

1.2 Contesto Applicativo e Obbiettivi

Il mondo della ricerca scientifica è caratterizzato da un vasto corpus di pubblicazioni, spesso dense e tecniche. Questi articoli scientifici seguono un formato standard: l'*articolo completo* con una dettagliata discussione dei metodi, risultati e conclusioni, e un *abstract* tecnico che mira a fornire una sintesi concisa dell'intero contenuto dell'articolo, permettendo ai lettori di

comprendere rapidamente l'obiettivo, la metodologia e i risultati principali del lavoro.

Tuttavia, c'è un crescente riconoscimento dell'importanza di rendere la scienza accessibile non solo agli esperti del settore, ma anche al grande pubblico. Mentre l'abstract tecnico è essenziale per gli specialisti, può risultare ostico per chi non è del settore. Ecco dove entra in gioco il concetto di "*lay summary*" o "*plain language summary*". Questi riassunti, come suggerito dal Centro KTDRR (usato, fra l'altro, come punto di partenza per la ricerca di articoli per il dataset), sono versioni semplificate di contenuti scientifici, tradotte in un linguaggio più accessibile, senza gergo tecnico, rendendo l'informazione chiara e comprensibile per tutti.

Nonostante l'importanza e la crescente richiesta di tali riassunti, nella maggior parte delle pubblicazioni scientifiche attuali manca un lay summary. L'elaborazione manuale di tali riassunti richiede tempo e competenze specifiche, e questo rappresenta una sfida significativa.

L'obiettivo principale di questa tesi è affrontare questa lacuna. Attraverso l'utilizzo di tecniche avanzate di intelligenza artificiale e di elaborazione del linguaggio naturale, si mira a sviluppare un sistema in grado di automatizzare il processo di sintesi di questi riassunti semplificati, partendo dal testo completo dell'articolo. Se riuscito, questo strumento potrebbe rivoluzionare il modo in cui la ricerca scientifica viene comunicata e compresa, rendendo la scienza più accessibile a un pubblico più ampio. Oltre a questo task, vorremmo generare autonomamente anche gli abstract.

Il NLP è una delle sfide più complesse in questo ambito, poiché richiede l'analisi lessicale e semantica del testo scritto. Negli ultimi anni, la ricerca in questo settore ha visto una crescente attenzione, soprattutto a causa della diffusione di dati testuali non strutturati. Tuttavia, questo tipo di dati presenta ambiguità e problematiche, che saranno analizzate in dettaglio.

La tesi si propone di valutare e confrontare vari metodi NLP e DL, evidenziando i loro vantaggi e svantaggi nell'ambito della sfida di estrarre conoscenze comprensibili dal pubblico da articoli scientifici biomedici. A supporto di questo obiettivo, viene presentato il nuovo dataset *SciLay*, appositamente sviluppato per questa ricerca. Questo dataset, ricavato dalla raccolta di articoli provenienti da oltre 50 riviste specializzate, include tra le sue caratteristiche campi designati a ospitare, in formato testuale semplice (e quindi facilmente elaborabile da un calcolatore), il testo integrale dell'articolo, il suo abstract e un riassunto lay, vale a dire in una forma semplificata e accessibile.

Il lavoro di tesi è stato suddiviso nei seguenti capitoli:

- **Capitolo 2** - Panoramica sulle varie tecnologie esistenti in letteratura utilizzabili per il problema posto.

- **Capitolo 3** - Analisi preliminare sulla struttura e modalità di raccolta dei dati.
- **Capitolo 4** - Introduzione alla modellazione del progetto con i primi approcci alle difficoltà riscontrate e alle relative soluzioni.
- **Capitolo 5** - Sviluppo concreto del progetto nelle sue varie fasi e relativo codice.

Capitolo 2

Le tecnologie disponibili

Nella successiva sezione, viene presentata una descrizione dettagliata degli strumenti utilizzati per realizzare gli obiettivi di questo progetto. Dopo un breve inquadramento del *Machine Learning*, ci concentreremo sul *Natural Language Processing*, dando particolare enfasi alle metodologie contemporanee più avanzate per la comprensione, generazione e sintesi del testo.

2.1 Machine Learning

Il Machine Learning, spesso abbreviato come ML, è un sottoinsieme dell'intelligenza artificiale che riguarda l'uso di algoritmi e modelli statistici per consentire ai computer di eseguire un compito senza l'uso di istruzioni esplicite. Invece, si basa su modelli che apprendono e si adattano alle informazioni e ai dati di input per prendere decisioni o effettuare previsioni senza essere esplicitamente programmati per farlo. Uno dei riferimenti più autorevoli in materia è Tom Mitchell, che ha fornito una definizione ben accettata di apprendimento automatico[3]:

> *Si dice che un programma apprende dall'esperienza E con riferimento ad alcune classi di compiti T e con misurazione della performance P, se le sue performance nel compito T, come misurato da P, migliorano con l'esperienza E.*

Questa definizione sottolinea l'idea centrale che, nel Machine Learning, un algoritmo migliora la sua performance nel tempo attraverso l'esperienza, che è tipicamente fornita sotto forma di dati di addestramento. Questo processo emula il *ragionamento induttivo* umano, essenziale per identificare tendenze o schemi, dando origine al concetto di *pattern recognition*. A differenza degli algoritmi classici, questi sistemi operano in modo probabilistico, implicando

che le relazioni identificate potrebbero non essere universalmente valide, ma piuttosto influenzate dai dati iniziali. La modalità di induzione più diffusa è la *generalizzazione*, che permette di dedurre informazioni riguardanti un intero insieme basandosi sull'analisi di una sua parte. Nello specifico i modelli di Machine Learning si basano proprio sul concetto di generalizzazione per portare a compito task, anche estremamente complicati da codificare esplicitamente.

2.1.1 La scelta dei dati

Quanto appena detto fa comprendere come il ruolo dei dati sia cruciale in questo tipo di sistemi. Maggiore è la quantità di dati e più tendenzialmente il modello sarà in grado di generalizzare correttamente e dunque fare previsioni corrette sottoponendo nuove istanze rispetto a quelle su cui si è addestrato. Questo concetto evidenzia anche la rinascita del Machine Learning degli ultimi due decenni. Col tempo, si è riconosciuta l'importanza vitale dei dati nella costruzione di modelli efficaci per supportare decisioni o effettuare previsioni in ambiti applicativi complessi. Di conseguenza, c'è stata una tendenza crescente alla raccolta e conservazione di grandi quantità di dati. Parallelamente, grazie all'avanzamento dell'hardware, che ha seguito la legge di Moore (anche se recentemente si sono manifestati segnali di un possibile rallentamento a causa delle limitazioni fisiche dei transistor), la capacità di elaborazione dei dati è cresciuta esponenzialmente. Questo ha reso possibile l'addestramento di modelli di Machine Learning sempre più sofisticati. Un aspetto da non sottovalutare è che quantità non coincide sempre con qualità. In altre parole, seguendo il principio "*Garbage In, Garbage Out*" (GIGO), dati inesatti, incompleti o distorti possono portare a risultati errati o fuorvianti, nonché a decisioni ingiuste o discriminatorie. Risulta pertanto cruciale prestare attenzione alla rappresentatività e all'equità durante la raccolta e la preparazione dei dati. Un noto esempio inerente è il caso *COMPAS*, riguardante un algoritmo utilizzato negli Stati Uniti per valutare il rischio di recidiva dei criminali. Esso ha sollevato molte preoccupazioni in merito alle questioni di equità e bias, soprattutto dopo che un'inchiesta da parte di ProPubblica che prese il nome di "*Machine Bias*"[4] ha suggerito che l'algoritmo potrebbe essere sbilanciato contro i detenuti afroamericani. In sintesi, i dati sono la colonna vertebrale del Machine Learning. La selezione, la preparazione e l'uso corretto dei dati determinano in gran parte il successo di qualsiasi progetto di ML. Una gestione attenta e riflessiva dei dati può fare la differenza tra un modello efficace e uno inefficace.

2.1.2 Perché Machine Learning?

Il machine learning rappresenta oggi uno dei pilastri fondamentali nell'ambito dell'intelligenza artificiale. Giganti tecnologici come Google, Microsoft e Netflix investono considerevoli risorse finanziarie nella ricerca di soluzioni innovative che possano elevare la qualità dell'esperienza degli utenti, traducendosi in un aumento dei loro profitti. La ricerca nel campo è in costante evoluzione, con la comunità scientifica che offre contributi preziosi in diversi settori:

- **Riconoscimento delle immagini**. Il ML, in particolare le reti neurali convoluzionali (CNN), ha rivoluzionato il riconoscimento e la classificazione delle immagini. Servizi come Google Photos e piattaforme di social media utilizzano il ML per riconoscere oggetti, volti e scene nelle immagini.

- **Elaborazione del linguaggio naturale (NLP)**. Il ML ha portato a progressi significativi nel riconoscimento vocale, nella traduzione automatica, nella generazione di testo e nell'analisi del sentimento. Siri di Apple, Google Translate e GPT-3 (e 4) di OpenAI sono esempi di applicazioni NLP basate su ML.

- **Raccomandazioni personalizzate**. Piattaforme come Netflix, Spotify e Amazon utilizzano algoritmi di ML per fornire raccomandazioni personalizzate agli utenti in base alle loro preferenze e comportamenti passati.

- **Veicoli autonomi**. Il ML è essenziale per le funzioni di guida autonoma, come il riconoscimento di ostacoli, la previsione del comportamento di altri utenti della strada e la decisione sul percorso da seguire.

Il vero valore aggiunto del machine learning, come già detto, risiede nella sua capacità di apprendimento e generalizzazione. L'incremento delle performance è spesso correlato alla quantità di dati disponibili, rendendo questi sistemi particolarmente adatti a gestire situazioni reali di elevata complessità, spesso al di là della capacità dei tradizionali algoritmi deterministici. La natura autoapprendente di queste tecniche permette di semplificare lo sviluppo di applicazioni, poiché l'algoritmo stesso si adatta autonomamente al contesto in cui opera.

2.1.3 Sviluppo di un modello

Come anticipato, ogni algoritmo di Machine Learning ha l'obiettivo di "imparare" dalle relazioni presenti nei dati forniti, in modo da poter "generalizzare",

ovvero effettuare previsioni accurate su nuovi dati mai visti prima. Occorre dunque definire matematicamente cosa significa trovare delle relazioni fra i dati. Formalmente siano $\mathbf{x} \in \mathbb{R}^n$ (con $n \geq 1$) un vettore di dati di input e un output $y \in \mathbb{R}$. Se vale

$$y = h(\mathbf{x}) = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n + \beta \tag{2.1}$$

possiamo affermare che esiste una qualche relazione tra $\mathbf{x}$ e y. La funzione h che lega le premesse ($\mathbf{x}$) alla conclusione (y) rappresenta un *modello*, e i coefficenti θ_i i suoi parametri. In sostanza, un modello è una rappresentazione matematica di un fenomeno, che cerca di descrivere le relazioni tra le variabili basandosi su osservazioni passate.

L'obiettivo principale nel Machine Learning è trovare un modello h che rappresenti al meglio le relazioni nel dominio di interesse. Formalmente, dunque, se l'equazione 2.1 lega input e output di una specifica istanza, qualora disponessimo di $m \geq 1$ istanze d'esempio, vorremmo determinare una funzione che riduca al minimo la discrepanza tra ogni output reale e l'output previsto dal modello. Questa quantità è definita *errore del modello*. Dato l'esempio $i-$esimo, siano y_i l'output reale e y_i' l'output previsto, allora trovare il modello migliore significa ottenere i parametri θ_i tali che si minimizzi l'errore di previsione. Poiché l'errore è funzione dei parametri, lo denoteremo come $E(\theta)$ dove θ è il vettore dei parametri. Allora avremo

$$\begin{aligned} E(\theta) &= \min \sum_{i=1}^{m} \frac{1}{2}(y_i - y_i')^2 \\ &= \min \sum_{i=1}^{m} \frac{1}{2}\left(y_i - h\left(\mathbf{x}_i\right)\right)^2 \end{aligned}$$

L'errore è rappresentato utilizzando la media dei quadrati degli errori (anche detto *errore quadratico medio* RMSE) per questioni di trattabilità matematica. La funzione in questione è differenziabile e pertanto è possibile l'uso di tecniche di ottimizzazione come la *discesa del gradiente* che tratteremo nella sezione successiva.

Esempio Quando abbiamo $n = 1$, ci troviamo in un contesto bidimensionale, con una dimensione dedicata alla variabile indipendente e l'altra alla variabile dipendente. Con $m \geq 1$ osservazioni disponibili, possiamo visualizzare il nostro modello h su un grafico cartesiano. Se le previsioni del modello si allineano lungo una retta, ci riferiamo a questo come *regressione lineare*. Il grafico potrebbe apparire come mostrato in figura 2.1, dove la retta $y = \theta x$ è stata calcolata in modo che il valore di θ minimizzi la somma dell'errore quadratico medio rispetto a tutti i punti (x_i, y_i) d'esempio.

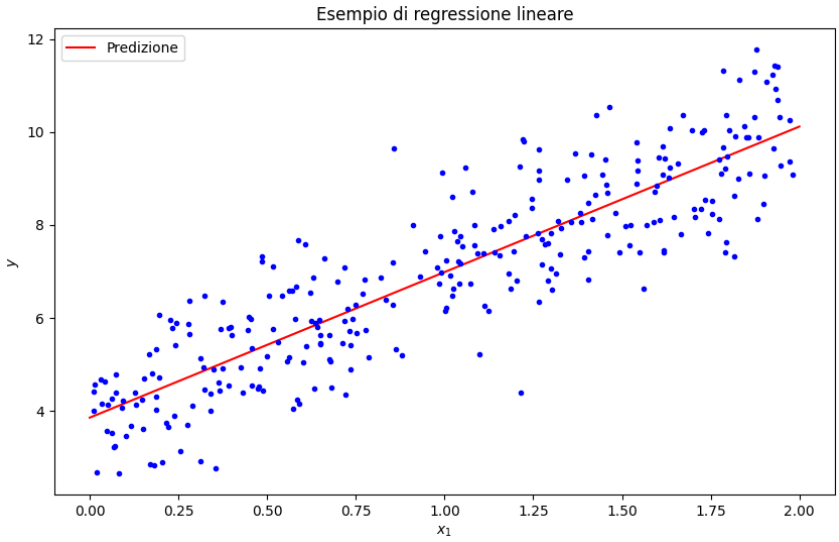


Figura 2.1: Esempio di regressione lineare.

2.1.4 Ricerca dei parametri: discesa del gradiente

Fino a questo momento abbiamo parlato di auto apprendimento della macchina in maniera del tutto generale, lasciando spazio alle ipotesi più fantasiose, fra cui l'idea che la macchina abbia un'effettiva intelligenza, simile a quella umana. In realtà, il modo in cui i sistemi auto apprendono è semplicemente quello di ricercare i parametri migliori (secondo quanto detto prima) per un determinato dominio applicativo. Ma come si trovano questi parametri? L'approccio più generale è chiamato *discesa del gradiente* che non è altro che un algoritmo di ottimizzazione utilizzato per minimizzare una funzione obiettivo attraverso aggiornamenti iterativi. Nel nostro caso la funzione obbiettivo da minimizzare sarà ovviamente $E(\theta)$, cioè l'errore quadratico medio sulla predizione. Essendo $E(\theta)$ una funzione, l'interpretazione geometrica di questo processo è di percorrere la curva lungo il punto in cui ha pendenza maggiore fino a trovare un minimo locale (o globale che sia). Il vettore direzione che ci indica verso dove avanzare alle iterazioni successive è chiamato, appunto, *gradiente*. Dal punto di vista matematico, poiché l'errore è funzione di θ, ovvero il vettore dei parametri dati alle variabili indipendenti, ci troveremo in uno spazio di $n+1$ dimensioni, dove n corrisponde al numero di parametri. Supponendo di avere $n = 2$ parametri graficamente avremo le situazioni mostrate in figura 2.2.

Formalmente, il gradiente corrisponde al vettore delle derivate parziali della funzione errore rispetto a ogni parametro. Se si ha un solo parametro, il gradiente si identifica con la derivata della funzione errore. Dal punto di vista matematico le derivate forniscono la pendenza istantanea di una curva

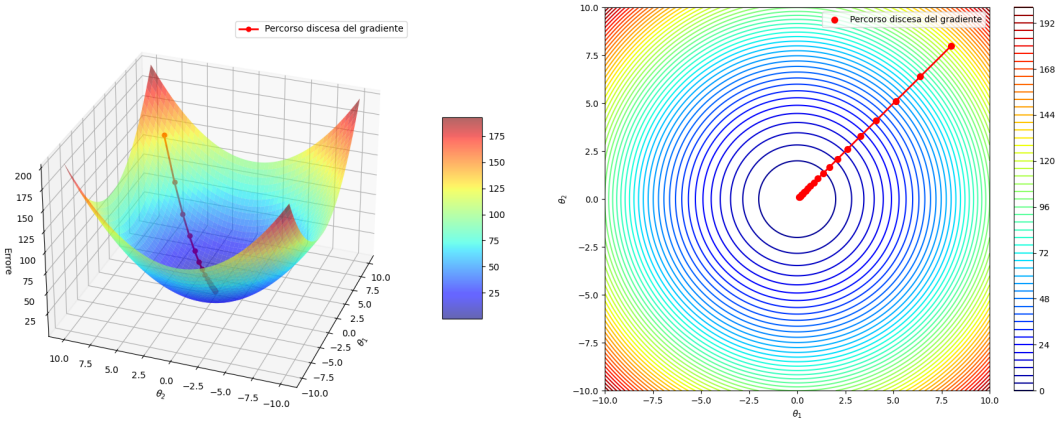


Figura 2.2: Rappresentazione grafica discesa del gradiente per un modello a due parametri.

in un qualsiasi punto essa sia calcolata (ammesso che sia sempre derivabile, cosa vera per $E(\theta)$ essendo MSE). Perciò, muovendosi nella direzione opposta al gradiente, si procede nella maniera più efficiente possibile, poiché in tale direzione la funzione decresce al tasso più veloce.

Considerando $E(\theta)$ come la funzione da minimizzare, il suo gradiente si calcola come

$$\nabla E(\theta) = \begin{bmatrix} \frac{\partial E}{\partial \theta_1} \\ \frac{\partial E}{\partial \theta_2} \\ \vdots \\ \frac{\partial E}{\partial \theta_n} \end{bmatrix}$$

A questo punto, per minimizzare E, aggiorniamo iterativamente il vettore dei parametri θ nella direzione opposta al gradiente. Definiamo come θ il vettore dei parametri all'iterazione precedente e θ' quello all'iterazione corrente. L'aggiornamento è dato da

$$\theta' = \theta - \alpha \nabla E(\theta) \tag{2.2}$$

dove α è un cosiddetto *iperparametro*, ovvero un parametro con cui regolare le modalità in cui il modello apprende. Questo determina l'ampiezza da percorrere ad ogni iterazione. Un α troppo grande potrebbe far "saltare" il minimo, mentre uno troppo piccolo potrebbe rendere la convergenza troppo lenta.

Il processo si ripete fino al soddisfacimento di un *criterio d'arresto*. Tale criterio può basarsi su diversi fattori:

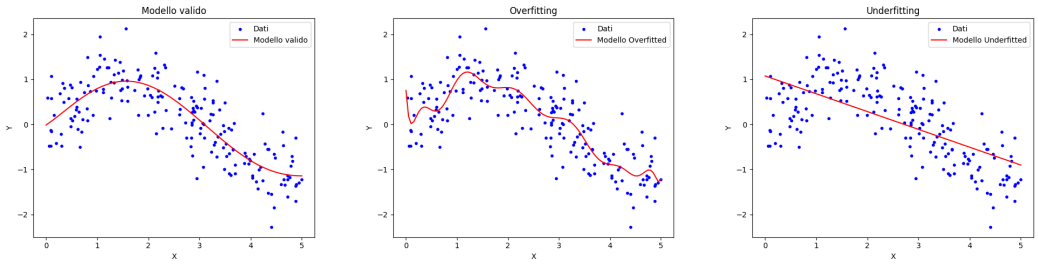
- Arresto quando l'errore scende sotto una soglia predeterminata.
- Arresto dopo un numero predefinito di iterazioni.

2.1.5 Validazione del modello

Per capire se il modello è effettivamente valido (da qui il nome *validazione*) a descrivere un certo dominio applicativo si possono utilizzare due differenti approcci.

1. Il metodo *hold-out* rappresenta una delle strategie più elementari e diffuse per determinare l'efficacia di un modello di apprendimento automatico. Consiste essenzialmente nella suddivisione dell'insieme di dati in due segmenti distinti: uno dedicato all'addestramento del modello e l'altro alla sua validazione. Queste due porzioni sono frequentemente denominate *train* e *test* set. È usuale destinare tra il 70-80% dell'insieme dei dati all'addestramento e il rimanente 20-30% alla validazione, anche se tali percentuali possono essere adattate in base al contesto e alla mole di dati a disposizione. Il set di addestramento è impiegato per istruire il modello. Al termine di tale fase, il modello è sottoposto a verifica utilizzando il set di test. Poiché questi dati non sono stati utilizzati durante la fase di addestramento, consentono di valutare le capacità del modello nell'affrontare dati inediti. È inoltre praticabile una segmentazione ulteriore dell'insieme dei dati, introducendo un *set di validazione*. Quest'ultimo è utilizzato per rifinire gli iperparametri del modello, mirando al miglioramento delle sue prestazioni. Conclusa questa ottimizzazione, il modello è nuovamente addestrato e, successivamente, la sua efficacia è testata con il set di test. È tuttavia essenziale considerare che l'affidabilità della valutazione potrebbe oscillare in base alla modalità di divisione dei dati. Se il set di test non risulta rappresentativo dell'intero insieme di dati, si potrebbero registrare incoerenze nella valutazione.

2. La soluzione ai limiti dell'hold out arriva dalla $k-$*fold cross validation* la quale mira a ottenere una valutazione più robusta e meno sensibile alle fluttuazioni che potrebbero derivare da una singola suddivisione casuale dei dati in set di addestramento e test, come avviene nel metodo hold-out. L'intero set di dati viene suddiviso in k sottoinsiemi (o "fold") di dimensioni approssimativamente uguali. La procedura di addestramento e test viene ripetuta k volte. Ogni volta, uno dei k fold viene utilizzato come set di test, mentre gli altri $k-1$ fold vengono combinati per formare il set di addestramento. Pertanto, ogni dato nel set viene utilizzato esattamente una volta come dato di test. In questo modo l'errore finale è dato dalla somma degli errori di previsione sui singoli test. Questo fa si che l'errore trovato sia più generale e meno influenzato dai dati scegli per addestrarlo.



(a) Modello rappresentativo dei dati.

(b) Modello affetto da overfitting.

(c) Modello affetto da underfitting.

Figura 2.3

Qualora a seguito della validazione del modello ci si rende conto che il modello non è in grado di generalizzare adeguatamente bene su nuovi dati non visti durante la fase di addestramento, possono presentarsi due problematiche (mostrate in Figura 2.3).

1. *Underfitting.* Si verifica quando un modello è troppo semplice per catturare la struttura sottostante dei dati. Di conseguenza, il modello potrebbe avere una cattiva performance sia sul set di addestramento che sul set di test. Le ragioni per cui questo accade possono essere svariate, ma le più frequenti sono la scelta di un modello troppo semplice per descrivere le relazioni fra dati, la presenza di troppi pochi parametri o ancora l'addestramento effettuato per troppe poche iterazioni.

2. *Overfitting.* Si verifica quando un modello è troppo complesso e inizia a "imparare" anche il rumore presente nei dati di addestramento, anziché la struttura sottostante dei dati stessi. In pratica, un modello che soffre di overfitting avrà ottime prestazioni sul set di addestramento, ma avrà prestazioni scarse sul set di test o su nuovi dati non visti.

Questo ci permette di capire che nel ML, di essenziale importanza è bilanciare la complessità del modello con la quantità e la qualità dei dati a disposizione, e monitorare attentamente le prestazioni del modello sia sul set di addestramento che sul set di test.

2.2 Reti Neurali

Le *reti neurali* rappresentano un pilastro centrale nel campo del machine learning. Sono modelli computazionali che traggono ispirazione dalla struttura e dalle funzionalità del cervello umano, progettati per replicare il modo in

cui gli esseri umani apprendono e processano le informazioni. Il parallelo con il cervello umano, seppure non stretto, serve come modello concettuale per comprendere i principi di base del loro funzionamento.

2.2.1 Struttura e Componenti di una Rete Neurale

Le reti neurali sono formate da unità di calcolo denominate, appunto, *neuroni* o *nodi*, raggruppate in strati, o *layers*. Questi si dividono in tre categorie principali:

- *Input layer*: rappresenta il punto di ingresso della rete, dove sono ricevuti gli input esterni.
- *Hidden layers* (strati nascosti): possono essere plurimi e, più sono numerosi, più la rete è in grado di gestire complessità, migliorando le proprie performance. Qui, i dati in ingresso sono elaborati ulteriormente.
- *Output layer*: è l'ultimo strato, attraverso il quale la rete fornisce i risultati dell'elaborazione dei dati in input.

Funzionamento di un Neurone

La Figura 2.4 illustra schematicamente il funzionamento di una singola unità neuronale. Ogni neurone riceve in input gli output di n neuroni connessi a esso. Ogni connessione i ha associato un *peso* w_i, ottimizzato durante la fase di addestramento del modello. Il neurone moltiplica ogni input x_i per il peso corrispondente, somma i prodotti ottenuti e aggiunge un *bias* b determinato anch'esso in fase di addestramento. Questa quantità è detta *somma ponderata degli input*. Questo bias è necessario per consentire al neurone di avere un grado di libertà aggiuntivo nel processo di apprendimento. Maggiore libertà comporta una maggiore adattabilità ai dati di addestramento. Questa operazione produce la *somma ponderata degli input*. Il bias fornisce al neurone un grado di libertà supplementare, permettendogli di adattarsi meglio ai dati durante la fase di apprendimento. La somma ponderata è poi processata da una *funzione di attivazione* che decide se il neurone deve essere attivato o meno, introducendo una componente di *non-linearità* essenziale per l'apprendimento di rappresentazioni complesse e relazioni non-lineari tra input e output. La formula appena descritta è la seguente:

$$\hat{y} = g(\mathbf{w} \cdot \mathbf{x} + b) = g\left(\sum_{i=1}^{n} w_i \cdot x_i + b\right)$$

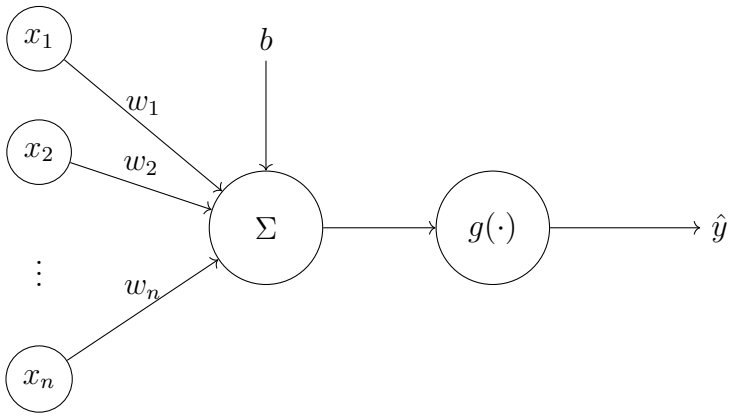


Figura 2.4: Schema di funzionamento unità neuronale.

Funzioni di Attivazione

Le funzioni di attivazione si classificano principalmente in due categorie, in base al loro codominio:

- *Funzioni di attivazione normalizzate*: queste funzioni, ad esempio, mappano i reali in intervalli del tipo $[0, 1]$ o $[-1, 1]$, risultando particolarmente utili nei layer di output quando la rete deve produrre probabilità, come nei task di classificazione. Alcuni esempi includono:
  - *Sigmoide.* Può essere utile nella classificazione binaria poiché è definita in $\mathbb{R} \to [0, 1]$. Si ottiene con
    $$\sigma(x) = \frac{1}{1 + e^{-x}}$$
  - *Tangente iperbolica.* Definita in $\mathbb{R} \to [-1, 1]$ e vale
    $$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$
- *Funzioni di attivazione continue*: queste mappano $\mathbb{R} \to \mathbb{R}$. e l'output determina non solo se il neurone è attivato, ma anche la quantità di segnale trasmesso. Un esempio noto è la *ReLU* (Rectified Linear Unit) ed è definita in $\mathbb{R} \to \mathbb{R}^+$. Se si attiva trasmette un segnale proporzionale alla somma pesata degli input, infatti è definita come
  $$\text{ReLU}(x) = \max(0, x)$$
  Se l'input (x) è negativo, il neurone restituisce 0, interpretato come uno "spegnimento" del neurone; se positivo, il neurone restituisce l'input ricevuto.

2.2.2 Processo di Addestramento

Durante l'addestramento di una rete neurale, come precedentemente menzionato, si scelgono i pesi delle connessioni tra neuroni e i bias di questi, in modo da rappresentare al meglio i dati di esempio. Analogamente agli algoritmi di Machine Learning (ML) tradizionali, l'accuratezza del modello viene valutata mediante una funzione d'errore, nota come *loss function*, che quantifica la divergenza del modello dalla soluzione desiderata. Il valore ottenuto, frequentemente denominato perdita" o errore", è indicativo dell'accuratezza del modello: valori bassi sono indice di predizioni precise, mentre valori alti denotano predizioni inesatte. L'obiettivo durante la fase di addestramento è minimizzare il valore della funzione di perdita. Per ridurre la loss, si adotta l'approccio della discesa del gradiente (vedi paragrafo 2.1.4). In particolare, ad ogni iterazione, comunemente definita come *"epoca di addestramento"* nel contesto delle reti neurali, si procede attraverso i seguenti passaggi:

- Si effettuano predizioni sui dati di esempio basandosi sui parametri attuali del modello.
- Si calcola la loss confrontando le predizioni con i valori reali.
- Si determina il gradiente della funzione di perdita rispetto ai parametri attuali del modello.
- Si aggiornano i parametri del modello muovendosi nella direzione che minimizza la funzione di perdita, come descritto dall'equazione 2.2.

2.2.3 Reti Neurali Feed-forward (FNN)

Le reti neurali di tipo *feedforward*, quali ad esempio i Multilayer Perceptron, rappresentano le forme più elementari e fondamentali di reti neurali. Questi modelli sono caratterizzati da un flusso d'informazione che procede in un'unica direzione, andando dal layer di input, passando per gli eventuali layer nascosti, fino a raggiungere il layer di output. In queste strutture, il processo attraverso il quale l'informazione è inviata da un neurone all'altro è conosciuto come *forwarding*. Nel dettaglio, in un'architettura feedforward, ogni neurone è interconnesso con tutti i neuroni presenti nel layer immediatamente successivo, formando una rete densamente connessa in cui l'informazione si muove in modo ordinato e strutturato da uno strato all'altro, senza cicli o ritorni. Questa struttura rigidamente organizzata si contrappone a modelli più complessi, come le reti *neurali ricorrenti* (trattate nel paragrafo successivo), che permettono connessioni in entrata anche da neuroni situati nello stesso layer o in layer successivi. Analizzando più profondamente il funzionamento di ogni singolo

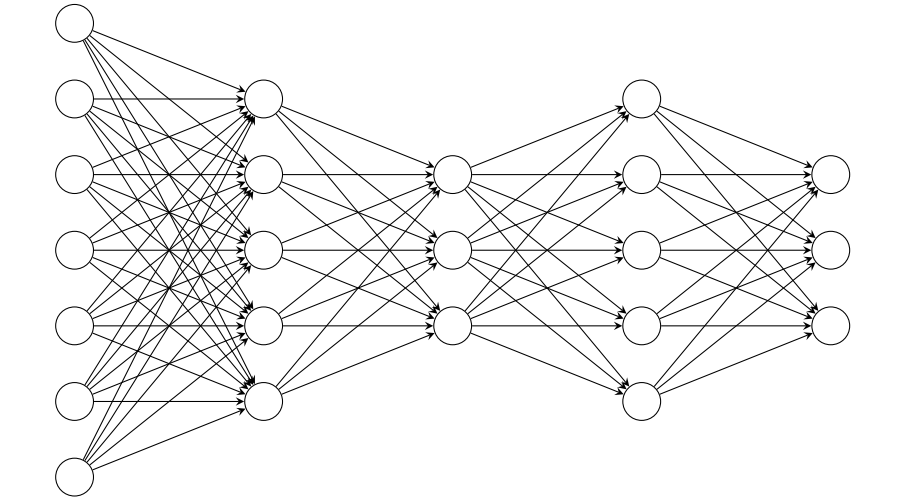


Figura 2.5: Feedforward Neural Network.

neurone, come illustrato nella Figura 2.4, è possibile notare che tutti gli input ricevuti da un neurone provengono esclusivamente da neuroni situati nel layer precedente, e ogni output prodotto è destinato unicamente ai neuroni del layer successivo. Ciò implica che ogni neurone opera come una sorta di stazione di transito, ricevendo segnali in ingresso, elaborandoli e inoltrandoli ai neuroni successivi, contribuendo così alla progressiva trasformazione dell'input iniziale fino all'ottenimento dell'output finale. Nelle reti neurali feedforward, i layer nascosti hanno il compito di elaborare e trasformare progressivamente le informazioni in ingresso, estraendo e combinando caratteristiche e rappresentazioni ad alto livello, indispensabili per l'esecuzione di compiti complessi. La presenza di più layer nascosti, ciascuno con un numero variabile di neuroni, consente alla rete di modellare relazioni e pattern sempre più complessi e astratti, aumentando la sua capacità di apprendimento e generalizzazione. La Figura 2.5 mostra un esempio di rete neurale feedforward, composta da un layer di input, tre layer nascosti e un layer di output. In questa rappresentazione grafica, è possibile osservare la strutturazione gerarchica e sequenziale della rete e l'organizzazione delle connessioni tra i diversi neuroni, che evidenzia la natura unidirezionale e strutturata del flusso informativo all'interno del modello.

2.2.4 Reti Neurali Ricorrenti (RNN)

Le *reti neurali ricorrenti* (RNN) sono una classe speciale di reti neurali che sono particolarmente efficaci nel trattare sequenze di dati, come serie temporali o sequenze di testo. A differenza delle reti neurali feedforward

(FNN), le RNN sono in grado di mantenere uno stato interno che cattura le informazioni relative alle parti precedenti della sequenza. In particolare le FNN presentano limitazioni significative quando si tratta di elaborare dati sequenziali o temporali (si pensi ad esempio a tutti quei task che coinvolgono testo dove per comprendere una frase è necessario avere contezza del significato delle parole precedenti in relazione a quella attuale). In particolare

- *Mancanza di memoria temporale.* Le FNN non hanno memoria interna o stato che può catturare l'ordine o la sequenza delle informazioni, rendendole inefficienti per problemi che richiedono la considerazione del contesto temporale o sequenziale. Le informazioni fluiscono senza lasciare traccia all'interno della rete. Ogni input è considerato a se stante.

- *Dimensione fissa dell'input.* Le FNN richiedono che l'input sia di dimensione fissa, il che è limitante per l'elaborazione di sequenze di lunghezza variabile. Infatti, come mostra Figura 2.5, quando si definisce l'architettura di una rete feedforward, si specifica il numero di neuroni in ogni layer, e quindi il numero di pesi in ogni layer è fisso una volta definita l'architettura.

- *Indipendenza delle feature.* Le FNN assumono che tutte le feature di input siano indipendenti l'una dall'altra, il che non è ideale per i dati sequenziali in cui l'ordine e il contesto delle informazioni sono cruciali.

Al contrario, le RNN, nascono con l'obbiettivo di superare questi limiti. Esse sono infatti capaci di *mantenere uno stato* e gestire sequenze di *dimensione variabile* di input/output.

Tutta la potenzialità di queste reti si basa su un principio molto semplice: piuttosto che usare i neuroni come aree di transito dell'informazione, può essere utile che ognuno di essi abbia la possibilità di mantenere in memoria uno *stato nascosto* (hidden state) che si aggiorni ad ogni nuova informazione in arrivo e che venga propagato assieme ad essa ai neuroni successivi. Questo permette alla rete di mantenere informazioni relative ai passaggi precedenti della sequenza mentre elabora ogni nuovo elemento. Potremmo in altre parole vedere questo stato come una "memoria" della rete. Questi stati nascosti vengono generati da delle *connessioni ricorrenti*, ovvero archi che formano cicli all'interno della rete: piuttosto che confluire in un'unica direzione, l'informazione può tornare anche indietro nella rete, per questo motivo si parla di *backpropagation.* In definitiva l'output di una rete neurale ricorrente è funzione sia del suo hidden state che dell'input fornito.

Formalmente occorre immaginare che le informazioni vengano processate ad intervalli discreti di tempo $t_0 \to t_1 \to t_2 \to \cdots$. Ad ogni iterazione t_i il vettore di stato nascosto viene aggiornato (dopo che l'informazione è

processata all'interno di un ciclo, o connessione ricorrente) mantenendo "in memoria" tutta la sequenza di input fino a quel momento. Durante la fase di *forwardpropagation* il neurone invia, oltre all'input arrivato al tempo t_i il suo stato nascosto aggiornato al passo t_{i-1}.

Reti di Elman

Le *reti neurali di Elman* sono il caso più semplice di RNN e dunque si prestano bene per comprendere il concetto di hidden state e connessioni ricorrenti. Una rete di Elman è molto simile ad una rete neurale feedforward a singolo strato nascosto, ma nello strato nascosto viene aggiunto un insieme di neuroni detti *unità di contesto*. In questo tipo di rete il comportamento appena descritto si ottiene in questo modo: per ogni neurone presente nello strato nascosto si aggiunge un'unità neuronale detta *di contesto* che riceve come input l'output del neurone nascosto corrispondente, e restituisce il proprio output allo stesso neurone. nascosto.

In altre parole la funzione 2.1, viene aggiornata nel seguente modo all'interno delle reti di Elman

$$\hat{y}_t = g(\mathbf{w} \cdot \mathbf{x}_t + \mathbf{u} \cdot \mathbf{h}_{t-1} + b)$$

La Figura 2.6 mostra come sono strutturate questo tipo di reti.

2.3 Natural Language Processing

L'elaborazione del linguaggio naturale (NLP, *Natural Language Processing*) è un campo interdisciplinare che combina linguistica, informatica e intelligenza artificiale, con l'obiettivo di permettere alle macchine di interpretare, comprendere e generare il linguaggio umano. Il linguaggio naturale è estremamente complesso e ricco di sfumature, implicazioni e ambiguità, il che rende il NLP una sfida difficile e al contempo estremamente affascinante.

2.3.1 Complessità del linguaggio naturale

Come già anticipato, il linguaggio naturale è intrinsecamente ambiguo e polisemico, ovvero una stessa parola può avere significati diversi a seconda del contesto in cui è inserita. Inoltre, il linguaggio è ricco di espressioni idiomatiche, metafore, sarcasmo e humor, che possono essere difficili da interpretare anche per gli esseri umani, figuriamoci per un calcolatore. Anche la struttura grammaticale e sintattica può variare enormemente tra lingue diverse, con differenze in ordine delle parole, morfologia, e costruzione delle frasi. A tutto questo si aggiungono

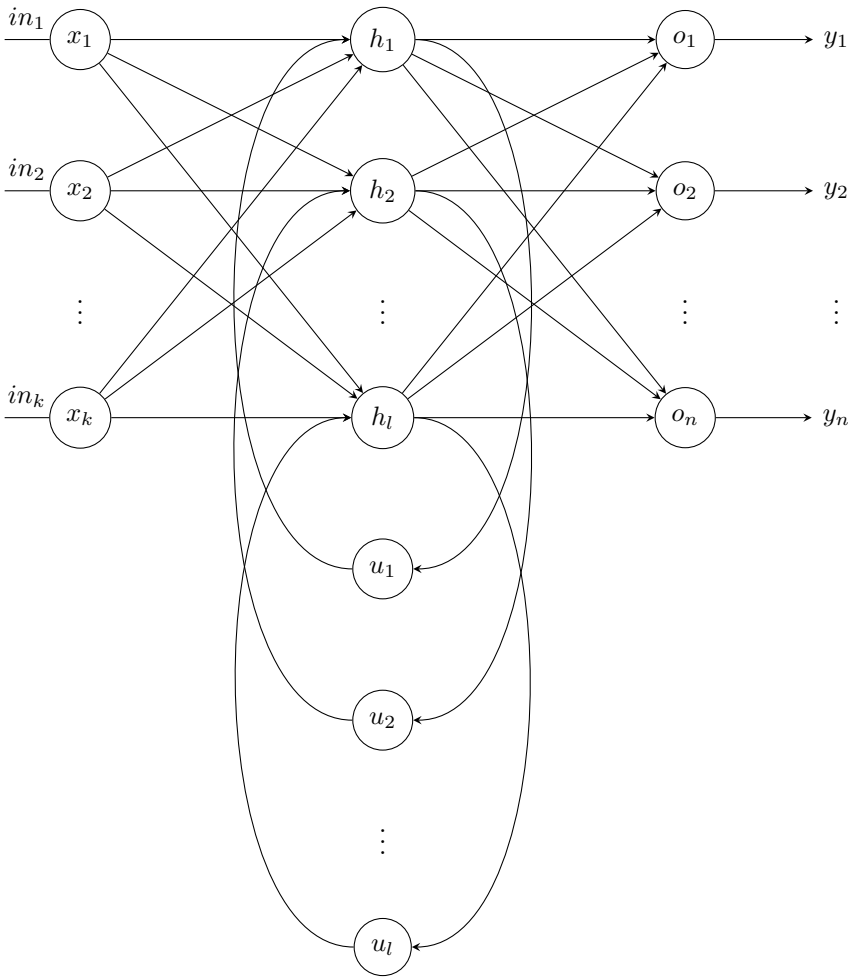


Figura 2.6: Rete neurale ricorrente di Elman.

anche la possibilità di abbreviare parole o intere frasi e di commettere errori sintattici o lessicali.

2.3.2 Sfide principali

In questo contesto così complicato e affascinante spiccano una serie di task ad oggi di particolare interesse.

- *Disambiguazione semantica delle parole* (WSD). Determinare il significato appropriato di una parola in un determinato contesto è una sfida fondamentale in NLP, data la presenza di parole identiche ma con significati diversi in base al contesto. Un esempio potrebbe essere il seguente:

  "Conseguire un titolo di studio avanzato è considerato un obiettivo *ambito* nel mondo accademico."
  oppure
  "L'*ambito* del Natural Language Processing è estremamente affascinante".

  Nelle due frasi il significato di *ambito* cambia totalmente.

- *Analisi del sentimento.* Comprendere il sentimento o il tono emotivo dietro una frase è cruciale in molte applicazioni, come il monitoraggio dei social media o la valutazione delle recensioni dei prodotti.

  "Il nuovo design di VirtuaLe mi piace" → Positiva.
  "Non so se il nuovo design di VirtuaLe mi piace" → Neutra.
  "Il nuovo design di VirtuaLe non mi piace" → Negativa.

- *Generazione di testo.* Creare testo coerente, grammaticalmente corretto e semanticamente sensato è un'importante sfida, utile in applicazioni come i chatbot e la creazione di contenuti.

- *Summarization.* Riguarda la produzione di una rappresentazione concisa delle informazioni principali contenute in un testo più esteso. L'obiettivo è di mantenere il significato e le informazioni cruciali del testo originale, riducendone però la lunghezza. Questo task è oggetto del seguente progetto di tesi.

Summarization estrattiva e astrattiva

Come accennato in precedenza, il task di summarization implica la creazione di una rappresentazione sintetica delle informazioni contenute in un documento più ampio, mantenendo inalterato il suo significato originale. Questa sfida,

oltre ad essere di grande interesse, è di fondamentale importanza per fornire agli utenti un assaggio di un documento, un articolo, una notizia o qualsiasi altro tipo di testo, permettendo loro di comprendere i punti chiave senza dover leggere l'intero contenuto.

Esistono principalmente due modalità di summarization: *extractive* e *abstractive*. Più nel dettaglio:

- *Extractive Summarization.* Questa metodologia crea riassunti individuando e estraendo le frasi o i segmenti di testo più significativi e informativi dal documento originale. Non viene creato nuovo testo, bensì vengono scelte porzioni del documento originale. Ciò richiede una significativa capacità di comprensione del testo per selezionare le sezioni più pertinenti, ma è meno esigente rispetto ai modelli abstrattivi. Gli algoritmi di extractive summarization frequentemente adoperano tecniche quali la pesatura delle parole, la posizione delle frasi nel documento, e la co-occorrenza di termini, che analizza la frequenza con cui parole o termini appaiono insieme in un determinato contesto, come un documento o una finestra di parole nel testo.

- *Abstractive Summarization.* Al contrario dell'approccio estrattivo, l'abstractive summarization produce nuove frasi che sintetizzano il contenuto principale del documento. Questo metodo necessita di una comprensione più profonda del testo e dell'abilità di riformulare concetti con nuove parole. Gli algoritmi di abstractive summarization fanno spesso ricorso a tecniche avanzate di deep learning, come i modelli di *attention* e i *Transformer* (discussi nei successivi paragrafi), per creare riassunti più coerenti e fluidi.

Come suggerisce il titolo di questa tesi, per i nostri esperimenti abbiamo deciso di utilizzare i modelli astrattivi disponibili in letteratura, piuttosto che tentare approcci estrattivi. Dato che l'obiettivo, come delineato nell'introduzione, è la generazione di riassunti lay, ovvero formulati in modo meno tecnico e più accessibile ad un ampio pubblico, nonché tecnici, il task ha portato con sé ulteriori sfide, quali l'identificazione delle sezioni cruciali del testo e la loro semplificazione. Per questa ragione, si è ritenuto che gli approcci extractive non fossero adeguatamente efficaci per raggiungere tale scopo.

2.3.3 Pre-processing di testo

Tutte le problematiche intrinseche del linguaggio delineate in precedenza vengono parzialmente arginate da una fase preliminare chiamata *pre-processing*. Il preprocessing del testo è un passaggio cruciale nell'ambito del Natural

Language Processing (NLP), il quale mira a semplificare il testo e ridurre la sua complessità, permettendo ai modelli di apprendimento automatico di operare più efficacemente. Ovviamente nel corso del tempo sono state introdotte un numero cospiquo di tecniche di questo tipo, ma le più importanti sono le seguenti:

- *Tokenizzazione.* La tokenizzazione è un passaggio cruciale nel preprocessing del testo per il Natural Language Processing (NLP). Questo processo consiste nella suddivisione del testo in ingresso (corpus) in unità più piccole e maneggevoli, denominate "*token*". Questi token rappresentano spesso parole, ma possono anche rappresentare simboli, frasi o altre unità di testo, a seconda delle specifiche necessità del task di NLP in questione. La tokenizzazione è essenziale per convertire il linguaggio naturale in una forma che può essere interpretata e manipolata dai modelli di machine learning, trasformando il testo grezzo in una struttura di dati più organizzata, come un vettore di token. Questo passo preliminare permette di preparare il testo per le successive fasi di elaborazione, come la vettorializzazione.

- *Rimozione delle Stop Words.* Nel linguaggio naturale esistono parole che non apportano alcun contributo al contenuto del testo, ma servono solamente ad arricchire il linguaggio e permettere una comprensione migliore a coloro che leggono. Esse sono parole funzionali che servono a collegare le parole contentive, cioè quelle che trasportano il significato principale del discorso, e a strutturare grammaticalmente le frasi. Senza le stop word, il linguaggio sarebbe frammentario e privo di coesione, rendendo difficile la comprensione del discorso. Per loro natura, dunque, non contribuiscono a dare significato al testo e pertanto non solo risultano inutili alla comprensione del testo da parte di un modello di NLP, ma aumentano la dimensionalità del problema rendendo più difficile e lungo l'addestramento. Alcuni esempi di queste parole possono essere ad esempio le congiunzioni come "e", "di", "in", ecc.

- *Lemmatizzazione e Stemming.* La lemmatizzazione e lo stemming sono tecniche di elaborazione del linguaggio naturale che mirano a ridurre le parole a una forma base o radice, facilitando così l'analisi del testo e migliorando l'efficacia dei modelli di NLP. In particolare:

  - La lemmatizzazione è il processo di riduzione di una parola alla sua forma canonica o lemma, cioè alla sua forma base che ha un significato nel dizionario. Questo processo tiene conto del contesto e

della morfologia della parola, utilizzando vocabolari e analisi grammaticali per assicurare che la parola risultante abbia un significato valido.

– Lo stemming, a differenza della lemmatizzazione, è un processo più rudimentale che riduce le parole alla loro radice tramite regole euristico-morfologiche, senza tenere conto del contesto. Questo processo può talvolta produrre radici che non hanno un significato valido nel dizionario. Sebbene potrebbe sembrare una pratica meno valida della lemmatizzazione in quanto può produrre parole prive di significato, tuttavia, poiché si tratta solamente di troncare la parola alla sua radice, il processo è estremamente pià efficiente e pertanto adatto a grandi moli di dati.

Entrambe le tecniche sono molto utili per ridurre la dimensionalità del corpus, in quanto tutte le parole derivate si riducono alla loro forma base in un caso e alla loro radice nell'altra.

- *Normalizzazione del Testo.* Riguarda la conversione del testo in una forma standard e uniforme, tipicamente rendendo minuscole tutte le lettere del testo. Questo può aiutare a migliorare le performance del modello.

- *Tagging Part-of-Speech.* Un modo per disambiguare, quanto meno in parte, il significato delle parole all'interno del testo è attribuire a ciascuna di esse un ruolotradeoff grammaticale all'interno delle varie frasi. Più nel dettaglio consiste nell'etichettare ogni parola in una frase con la sua parte del discorso grammaticale, come sostantivo, verbo, aggettivo, avverbio, ecc. Questo processo è essenziale per comprendere la struttura grammaticale delle frasi e per analizzare le relazioni semantiche tra le parole.

- *Creazione di N-grammi.* Non è sempre possibile attribuire un significato autonomo a ogni parola all'interno di un testo. Esistono infatti termini composti, come "New York", che possono essere frazionati in modo errato durante la tokenizzazione. Per ovviare a questo inconveniente, è possibile ricorrere alla creazione di n-gram, ossia tuple composte da n parole consecutive, con n arbitrario. È importante sottolineare che, all'aumentare di n, la probabilità di formare combinazioni di parole coerenti e sensate tende a diminuire significativamente. Le modalità di POS tagging sono molteplici fra cui approcci statistici e modelli di RNN addestrati a capire le relazioni fra parole su centinaia di migliaia di frasi annotate.

- *Rappresentazione Vettoriale (Vectorization).* La vettorializzazione dei token è un processo cruciale che converte parole, frasi o documenti, rappre-

sentati come token, in vettori di numeri. Questo processo è fondamentale poiché i modelli di machine learning e le tecniche di analisi dei dati possono lavorare efficacemente solo con dati numerici. La rappresentazione vettoriale permette di quantificare e analizzare il testo in modo che i modelli di apprendimento possano comprenderlo e processarlo. Esistono diverse strategie per la vettorializzazione, che spaziano da metodi più semplici, come il modello *Bag-of-Words (BoW)*, dove il documento è descritto come un vettore le cui componenti rappresentano la frequenza delle parole nel documento, a metodi più sofisticati come i *Word Embeddings*. In questo caso esistono molti algoritmi, fra cui *Word2Vec*[5] e *GloVe*[6], tutti quanti accumunati dal concetto di rappresentare le parole come vettori densi in uno spazio vettoriale continuo. In questo modo si è in grado di catturare la semantica delle parole, consentendo a parole semanticamente simili di avere vettori simili, graficamente questo è mostrato in Figura 2.7.

L'utilizzo di tali rappresentazioni vettoriali offre il considerevole vantaggio di poter applicare operazioni matematiche ai vettori, ottenendo risultati semanticamente coerenti nelle parole corrispondenti. Alcuni esempi celebri di questa proprietà sono dati dalle seguenti operazioni vettoriali:

$$\text{King} - \text{Man} + \text{Woman} = \text{Queen}$$
$$\text{Human} - \text{Animal} = \text{Ethics}$$

In questo modo, il modello può effettivamente "ragionare" in termini semanticamente significativi, aprendo nuove possibilità nella modellazione del linguaggio naturale e nell'analisi del testo.

2.4 Tecniche astrattive allo stato dell'arte

Alcune delle architetture più avanzate, tecnicamente definite *allo stato dell'arte*, per compiti quali la generazione di testo, la comprensione del linguaggio e la traduzione, come i seguenti modelli *BART* [7], *T5* [8] e *GPT-3* [9] (quest'ultimo Large Language Model o LLM[1]), si basano sulle tecniche introdotte di seguito fra cui spiccano:

[1]Nella letteratura corrente, non esiste una definizione formale universalmente riconosciuta per i Large Language Models (LLM). Tuttavia, è generalmente accettato nella comunità scientifica che i modelli di linguaggio con oltre un miliardo di parametri rientrano in questa categoria.

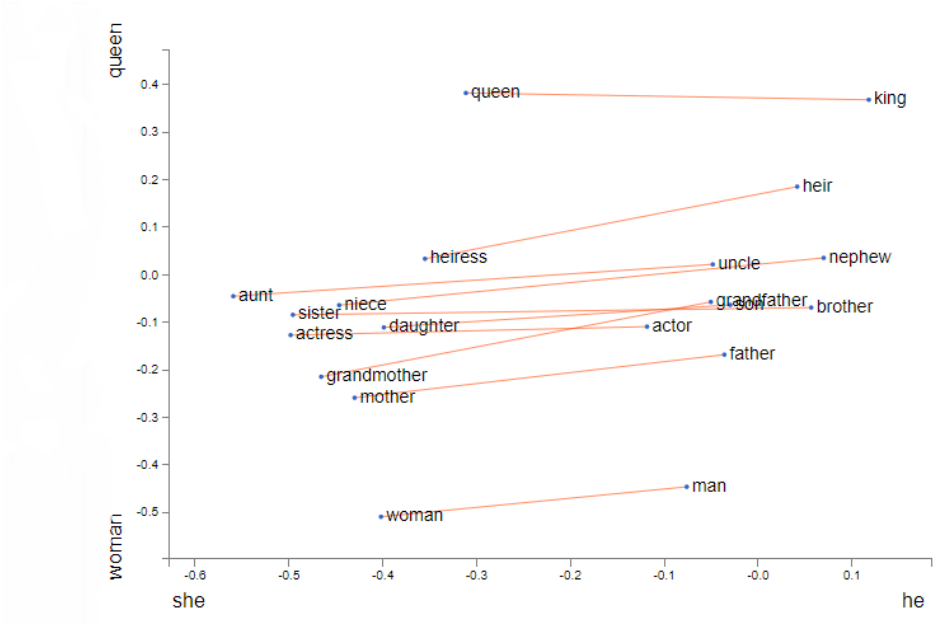


Figura 2.7: Proiezioni di parole su spazio vettoriale $\mathbb{R}^2$.
Fonte: https://lamyiowce.github.io/word2viz/

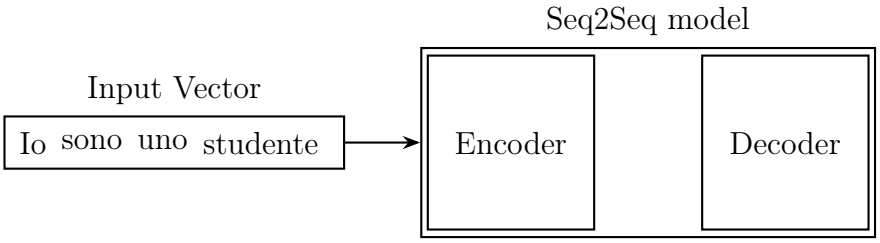
- Modelli *Sequence-to-Sequence*.
- Meccanismo di *Attention*.
- Architettura *Transformers*.

2.4.1 Modelli Seq2Seq

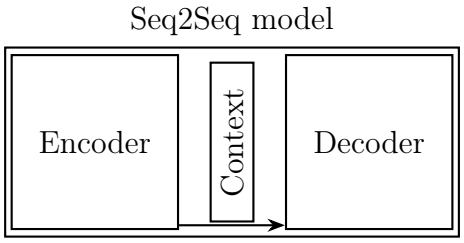
I modelli *Seq2Seq* (Sequence-to-Sequence) sono un tipo di architettura di rete neurale utilizzata per la generazione di sequenze, in cui una rete prende in input una sequenza e produce in output un'altra sequenza. Questo tipo di modelli è stato ampiamente utilizzato in una varietà di applicazioni, tra cui la traduzione automatica, la generazione di testo, la sintesi vocale e molto altro, nella pubblicazione originale [10] erano basati su RNN, ma con il passare del tempo si sono evoluti fino a portare, come anticipato, alle architetture più moderne che fanno uso di transformers e self-attention.

Funzionamento

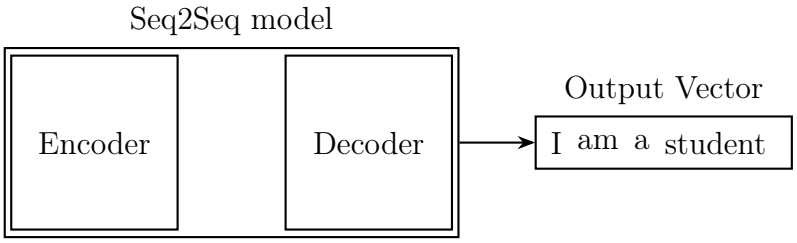
Come suggerisce il nome, seq2seq è un modello che prende in input una sequenza di oggetti (parole, lettere, pixel di un'immagine, ecc.) e restituisce in output una nuova sequenza di oggetti, tipicamente correlati in qualche modo ai



(a) Ingresso delle parole all'interno dell'encoder.



(b) Creazione e forwarding del context vector.



(c) Produzione delle parole di output da parte del decoder.

Figura 2.8

primi. Ad esempio se si stesse implementato un modello di traduzione, l'input sarebbe una frase in una lingua e l'ouput sarebbe la sua traduzione in un'altra.

Come mostra la Figura 2.8, questi modelli sono composti da due componenti

- *Encoder*. Questa prima parte si occupa di processare ciascun elemento nella sequenza di input, generare un vettore chiamato *context*, che catturi le informazioni della sequenza, e di spedirlo al *decoder*.
- *Decoder*. Posto a cascata dopo l'encoder, questo si occupa, una volta che il primo gli ha inviato il context, di produrre la nuova sequenza di output elemento per elemento, esattamente come l'encoder ha consumato quella in input.

In particolare il vettore di contesto è un un array di numeri decimali come indicato in Figura 2.9.

0.11
0.03
0.81
-0.62

Figura 2.9: Esempio di context vector.

La sua dimensione dipende strettamente dalla complessità del modello in quanto, per poterlo ricevere correttamente come input, l'encoder avrà un numero di unità nascoste pari a questa dimensione. Nelle applicazioni reali si utilizzano context vectors di dimensione 256, 512 o 1024.

Seq2Seq con RNN

Come anticipato, i primi modelli seq2seq sono stati concepiti per essere implementati mediante reti neurali ricorrenti (si veda paragrafo 2.2.4). Nello specifico sia encoder che decoder sono pensate per essere RNN, pertanto entrambe richiedono due input ad ogni iterazione:

- L'input in senso stretto. Nel caso dell'encoder corrisponde alla rappresentazione vettoriale di ciascuna parola dell'input.
- Uno stato nascosto che servirà a entrambe le componenti del modello per avere una comprensione più profonda del significato dell'intera frase passata. Come vedremo in seguito l'hidden state generato dall'ultima iterazione dell'encoder rappresenta proprio il vettore di contesto fornito in input al decoder.

Nel concreto, supponendo di percorre passo passo l'esecuzione di un modello Seq2Seq implementato con RNN, su un task di traduzione dall'italiano all'ingelse della frase "sono uno studente" avremo i seguenti passaggi

[Iter 1] Il primo elemento della frase in input viene convertito nella sua forma vettoriale e fornito all'encoder. Questo produce il primo Hidden state dell'encoder, che chiameremo $\mathbf{h}_1$.

[Iter 2] La seconda parola viene convertita, fornita all'encoder, il quale, combinandola con $\mathbf{h}_1$, genera il secondo hidden state $\mathbf{h}_2$.

[Iter 3] Alla terza e ultima iterazione sulla sequenza di input la terza rappresentazione vettoriale viene combinata con $\mathbf{h}_2$ per generare $\mathbf{h}_3$. Poiché la sequenza in input si è esaurita, questo hidden state corrisponde al context vector citato in precedenza. Questo viene passato al decoder.

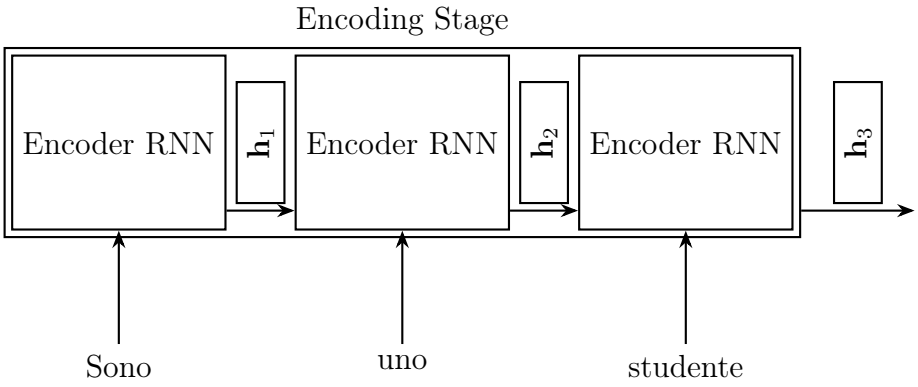


Figura 2.10: Schema di funzionamento della fase di encoding.

[Iter 4] Il decoder usa il context vector per generare un Hidden State e la prima parola della frase di output. L'hidden state viene usato, assieme alla prima parola generata, per produrre la seconda parola e il secondo hidden state. Il procedimento si ripete fino al termine della frase.

In Figura 2.10 è schematizzato quanto detto nei punti precedenti (la fase di decoding, eseguendo simmetricamente, non è stata rappresentata).

2.4.2 Attention

Una delle sfide principali dei modelli Seq2Seq basati su RNN è la loro incapacità di gestire efficacemente sequenze di parole eccessivamente lunghe. Questo è dovuto alla presenza del context vector, che agisce come un collo di bottiglia. Teoricamente, il context vector dovrebbe contenere tutte le informazioni necessarie dalla sequenza di input per permettere al decoder di generare la sequenza di output corretta. Tuttavia, la realtà mostra che condensare informazioni di una lunga sequenza in un vettore statico di dimensione fissa è complesso e conduce a una perdita di informazioni significativa. Il meccanismo di *Attention* [11], [12], è stato concepito proprio per superare questa limitazione, permettendo al modello di focalizzarsi su porzioni pertinenti della sequenza di input durante la generazione di ogni parola in output.

La Figura 2.11, ad esempio, illustra una heatmap che evidenzia come il modello (il cui task è di tradurre dall'inglese al francese) sia capace di non allineare semplicemente la prima parola in input con la prima in output, ma di focalizzarsi dinamicamente su segmenti di input rilevanti per la sintesi di ogni singola parola in output.

Più precisamente, questa heatmap correla ogni parola in input con ogni parola in output, formando una griglia dove la cella ij si illumina con un

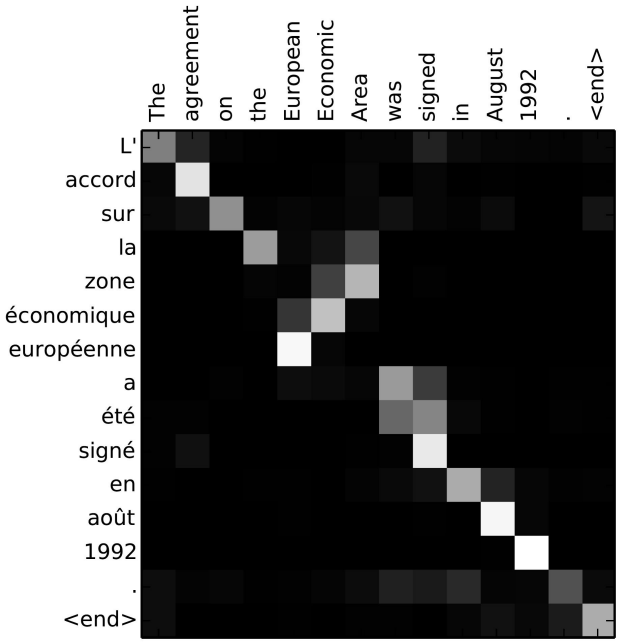


Figura 2.11: Heatmap di Attention per task di traduzione.
Fonte:
https://github.com/tensorflow/nmt/blob/365e7386e6659526f00fa4ad17eefb13d52e3706/nmt/g3doc/img/attention_vis.jpg

colore più luminoso quanto più la parola j in input è legata alla parola i in output. È interessante notare che, in generale, le celle lungo la diagonale principale sono più luminose, rispecchiando la tendenza a mantenere l'ordine delle parole, anche tra lingue differenti. Tuttavia, osservando la sequenza "European Economic Area" è evidente come, a causa dell'inversione dell'ordine delle parole in francese ("européenne économique zone"), la correlazione più forte si discosti dalla diagonale principale.

Funzionamento

Un modello che fa uso di attention si discosta dai classici modelli Seq2Seq RNN in due modi principalmente.

In primo luogo, a differenza di quanto succedeva prima, l'encoder passa *tutti* i vettori di stato nascosto al decoder prodotti dalla consumazione progressiva delle parole in input. Cioè lo schema in Figura 2.10 diventa come quello mostrato in Figura 2.12.

In secondo luogo, il decoder, impiega una procedura ulteriore, chiamata appunto *attention* a ogni step, operando nel seguente modo per concentrarsi sulla porzione pertinente della frase in input:

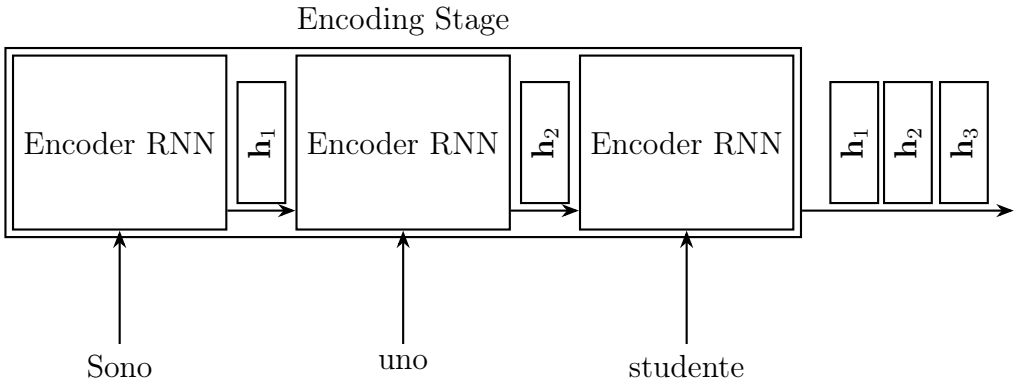


Figura 2.12: Schema di funzionamento della fase di encoding in un modello di attention.

1. Analizza tutti gli hidden state ricevuti dall'encoder. Ognuno di questi hidden state è prevedibilmente più strettamente associato alla parola dell'input che l'encoder ha processato nel corrispondente step; così, come mostrato in Figura 2.12, l'hidden state $\mathbf{h}_2$ è prevalentemente correlato alla seconda parola, $\mathbf{h}_3$ alla terza, e così via. Inoltre, è necessario l'hidden state del decoder nello step corrente.

2. Assegna ad ogni vettore uno score in relazione allo stato nascosto del decoder all'iterazione t corrente (che chiameremo $\mathbf{s}_t$). Questo score può essere calcolato in diversi modi. Le due modalità seguenti sono le più note:

   (a) *Additive Attention.* Si calcola con

   $$\text{score}(\mathbf{h}_i) = \mathbf{v}_a^T \tanh\left(\mathbf{W}_a[\mathbf{s}_t; \mathbf{h}_i]\right)$$

   dove il ; rappresenta la concatenazione dei vettori e $\mathbf{v}_a, \mathbf{W}_a$ sono dei parametri che il modello apprende in fase di training. Fra le due questa risulta essere la più espressiva, cioè è in grado di modellare relazioni più complesse e non lineari tra gli stati nascosti grazie all'uso della funzione di attivazione tanh, tuttavia è estremamente costosa da calcolare.

   (b) *Multiplicative Attention.* Si calcola con

   $$\text{score}(\mathbf{h}_i) = \mathbf{s}_t^T \mathbf{W}_a \mathbf{h}_i$$

   Questa formula ha diversi pregi fra cui l'efficienza computazionale (richiede solo operazioni di prodotto scalare e matriciale) e semplice. Tuttavia è intrinsecamente lineare, e questo può limitare la potenzialità di catturare relazioni complesse nei dati.

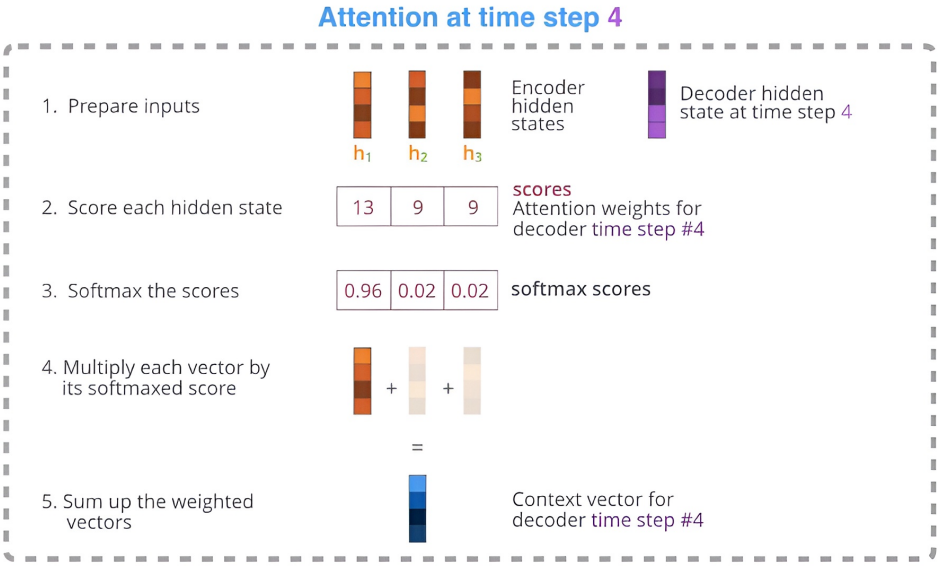


Figura 2.13: Schema di funzionamento meccanismo di attention.
https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

3. Successivamente, si applica la funzione softmax su tutti gli score, che amplifica gli hidden state con score elevati e attenua quelli con score bassi (in pratica, la softmax genera un valore più alto per gli stati con score maggiore e uno più basso per quelli con score minore).

4. Ogni vettore è quindi moltiplicato per il rispettivo score (a cui è stata applicata precedentemente la softmax) e i risultati sono sommati, formando così il context vector del decoder per l'iterazione attuale.

In sintesi, se ci troviamo, ad esempio, al quarto step del processo, dopo aver generato i tre vettori di stato nascosto nell'encoder nei primi tre step e averli trasmessi al decoder, la generazione del primo context vector nel decoder (utilizzato, come nei modelli seq2seq RNN, per generare la prima parola in output) avverrà come rappresentato in Figura 2.13.

Il primo passaggio consiste nella preparazione dei vettori di input, ossia gli hidden state $h1, h2, h3$ ell'encoder e l'hidden state del decoder al quarto passo. Successivamente, si calcolano gli score dei tre hidden state dell'encoder e si applica la softmax su di essi per enfatizzare i valori degli score più alti e ridurre quelli più bassi. Si moltiplica la softmax per ogni hidden state e poi si sommano insieme. Questo equivale a:

$$\text{context}_n = \sum_{i=1}^{n-1} (\text{Softmax}(\text{score}(\mathbf{h}_i)) \cdot h_i)$$

dove il prodotto è da intendersi come prodotto scalare (cioè i termini si moltiplicano uno ad uno e poi si sommano fra loro).

Per sintetizzare il processo eseguito quando si fornisce una stringa, ad esempio "Sono uno studente", a un modello Seq2Seq con Attention per un compito di traduzione, i seguenti passaggi vengono eseguiti:

1. Al termine della fase di encoding, l'encoder trasmette al decoder i tre hidden state, denotati come $(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3)$, generati durante l'elaborazione delle tre parole che compongono la stringa di input.

2. Nella quarta iterazione (considerando le tre iterazioni precedenti eseguite dall'encoder), il decoder riceve in input l'embedding della stringa speciale di terminazione <END> e uno hidden state iniziale del decoder, denotato come $\mathbf{h}_{\text{init}}$ (si ricorda che anche il decoder è una RNN, pertanto ad ogni iterazione prenderà in input uno stato nascosto e un embedding).

3. Attraverso il meccanismo di attention, precedentemente definito, si determina un context vector, $\mathbf{c}_4$, che viene concatenato all'hidden state $\mathbf{h}_4$ prodotto dal decoder RNN a seguito della computazione sull'hidden state iniziale $\mathbf{h}_{\text{init}}$ e la stringa di terminazione <END>.

4. La concatenazione $[\mathbf{c}_4; \mathbf{h}_4]$ è successivamente inoltrata a una rete neurale feed-forward che genera la parola per il passo corrente.

Successivamente, la parola generata al passo 4 è utilizzata dal decoder come input per la quinta iterazione. In questo passaggio, il decoder elabora la parola e l'hidden state 4 in input, generando un nuovo hidden state, $\mathbf{h}_5$ che verrà concatenato al vettore di contesto $\mathbf{c}_5$ calcolato come prima. La concatenazione $[\mathbf{c}_5; \mathbf{h}_5]$ fiene passato alla FNN che produce la seconda parola di output. viene poi fornita alla rete neurale feed-forward che produce la seconda parola in output. Il procedimento si ripete in modo analogo anche per la produzione della terza parola. L'intero processo è illustrato schematicamente nella Figura 2.14.

2.4.3 Transformers

Architettura formalmente presentata nel paper intitolato "Attention is All You Need" [13], con l'obiettivo di superare i modelli sequence-to-sequence basati su RNN e meccanismi di Attention, dettagliati nei paragrafi precedenti.

Il titolo del paper suggerisce la novità cardine introdotta dagli autori: una nuova architettura, denominata *transformer*, che si fonda esclusivamente sul principio dell'attention, eliminando la necessità di impiegare reti neurali

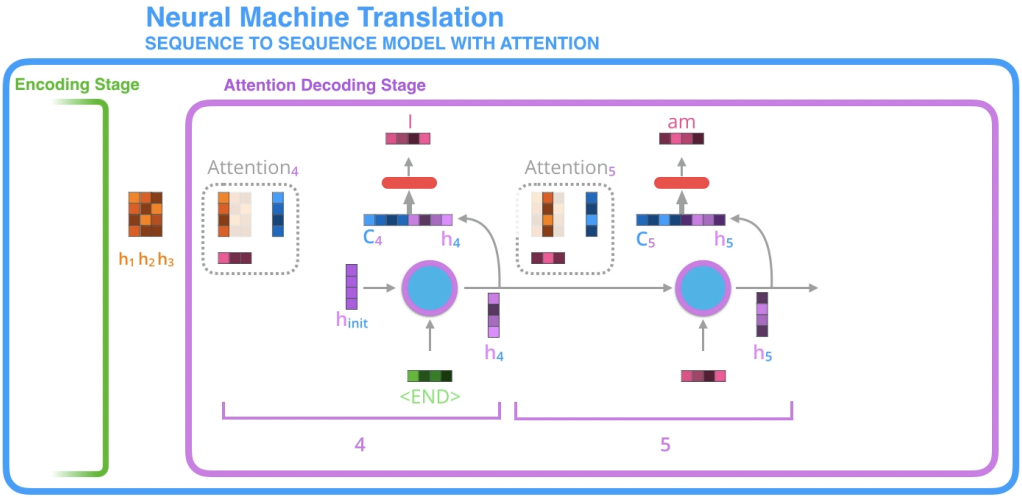


Figura 2.14: Schema di funzionamento di un decoder con attention.
https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

ricorrenti. Questo approccio ha dimostrato non solo la sua validità, ma ha anche permesso di raggiungere un significativo miglioramento delle performance nei compiti di elaborazione del linguaggio naturale.

Funzionamento

L'architettura basata su Transformers, come sarà illustrato, mantiene una struttura non dissimile da quella dei modelli sequence-to-sequence (seq2seq) implementati con Reti Neurali Ricorrenti (RNN). Specificatamente, i modelli seq2seq sono delineati da una struttura encoder-decoder. In tale configurazione, l'encoder è responsabile per trasformare una sequenza di rappresentazioni simboliche (i.e. parole)

$$(x_1, \ldots, x_n)$$

in una corrispondente sequenza di rappresentazioni continue (gli stati nascosti),

$$\mathbf{H} = (\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n)$$

Una volta ottenuto $\mathbf{H}$, il decoder genera sequenzialmente una serie di simboli (parole)

$$(y_1, y_2, \ldots, y_m)$$

un elemento alla volta. Ad ogni passo, il modello, essendo auto-regressivo, impiega i simboli precedentemente generati come input aggiuntivo per la

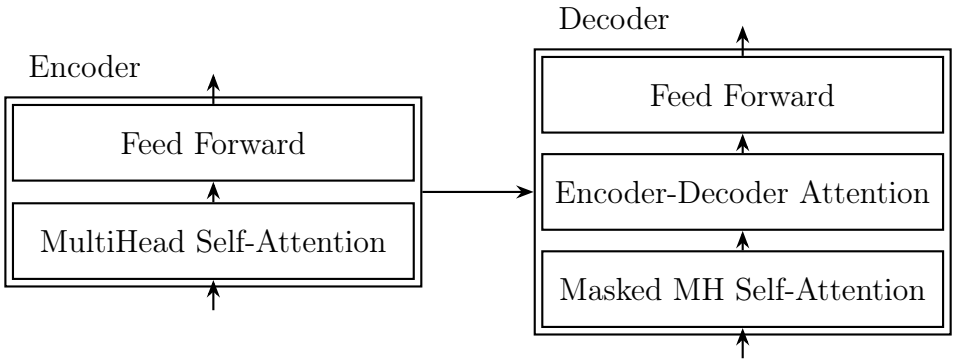


Figura 2.15: Architettura di base di un Transformers

produzione del simbolo successivo, consumando, quindi, i simboli prodotti in combinazione con lo stato nascosto precedente per generare la sequenza di output.

I Transformers sono strutturati seguendo un principio analogo, caratterizzati da $N = 6$ layer di *encoder* concatenati e altrettanti layer di *decoder*, disposti in maniera analoga.

Quello che cambia è il contenuto di decoder ed encoder. Come mostra figura 2.15, avremo

- L'input dell'*encoder* attraversa in quest'ordine un layer di *self attention* (definita nel dettaglio in seguito), che consente all'encoder di capire come codificare la parola osservando la relazione che questa ha con le altre e una rete feed forward (FNN).

- Il *decoder* è fatto in maniera analoga, avendo in più fra la FNN e il layer di self attention, un layer di attention come è stata definita nel capitolo precedente, che gli consente di concentrarsi sulle parti rilevanti della sequenza di input per produrre l'output.

Self-Attention La *self-attention* opera, a livello concettuale, in maniera analoga alla meccanica dell'attention delineata nel precedente paragrafo, permettendo al modello di discernere le relazioni semantiche intrinseche fra le parole presenti nella sequenza di input. Ciò che distingue la *self-attention* è la metodologia attraverso cui vengono calcolati gli score, utilizzati successivamente per generare i vettori di contesto.

Considerando la frase:

"The animal didn't cross the street because it was too tired"

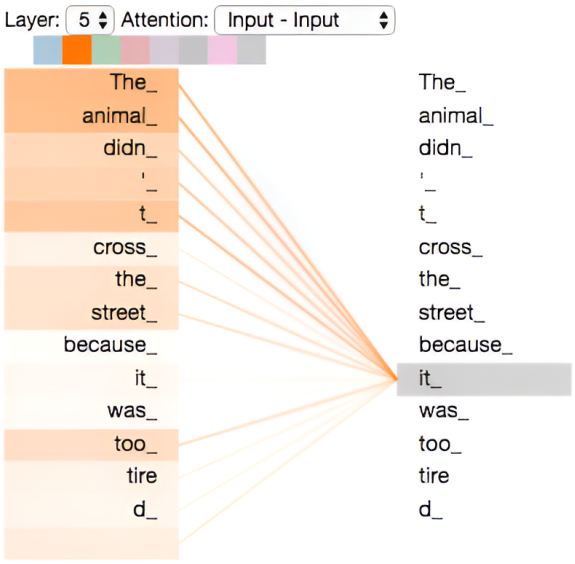


Figura 2.16: Visualizzazione grafica della self-attention per una frase.

la *self-attention* consente al modello di associare correttamente il pronome "it" con "animal" piuttosto che con "street". Questa correlazione è illustrata graficamente in Figura 2.16, dove l'intensità cromatica di una parola denota il grado di correlazione con la parola "it".

Il procedimento operativo della *self-attention* può essere suddiviso nei seguenti passi:

1. Inizialmente, da una matrice di embeddings di input, denotata come $\mathbf{X}$, (una riga per ogni embedding), vengono generate tre matrici distinte: matrice delle *query* $\mathbf{Q}$,matrice delle *chiavi* $\mathbf{K}$ e matrice dei valori $\mathbf{V}$. Ogni riga delle suddette matrici corrisponde a un embedding distinto, calcolato mediante la moltiplicazione della matrice $\mathbf{X}$ per delle matrici di pesi $\mathbf{W^Q}, \mathbf{W^K}, \mathbf{W^V}$ apprese durante la fase di addestramento, come illustrato in Figura 2.18.

2. Successivamente, vengono calcolati gli score di correlazione per ogni parola nella sequenza di input, con ciascuna delle altre parole presenti, generando n^2 score, ove n rappresenta il numero di parole nella sequenza. Gli score di attention sono ottenuti attraverso i seguenti passaggi:

   - Ogni riga di $\mathbf{Q}$ viene moltiplicata per ogni riga di $\mathbf{K}$ (nel concreto $\mathbf{K}$ verrà trasposta per consentire questo prodotto).
   - Il risultato ottenuto viene normalizzato dividendo per un *fattore di scala* $\sqrt{d_k}$ che non è altro che la dimensione della matrice delle

chiavi. Questo serve perché la dimensione delle chiavi (e anche di query) sarà proporzionale al numero di elementi in input perciò la moltiplicazione di ogni query per ogni chiave potrebbe portare a valori molto grandi se l'input è esteso. Questo passaggio è fatto al fine di mitigare l'incremento dei valori derivante da sequenze di input ampie.

- Si procede applicando la funzione softmax a ciascuno degli score ottenuti. Tale operazione ha lo scopo di quantificare il grado di rilevanza della parola i-esima in posizione j. È naturale aspettarsi che, quando $i = j$ lo score risultante tenda ad essere più elevato. L'obiettivo di questa procedura è preservare l'integrità delle parole di interesse, mentre si attenua l'importanza di quelle considerate irrilevanti per la generazione del segmento di output desiderato.
- Infine, ciascun elemento ottenuto viene moltiplicato per ogni elemento della matrice dei valori $\mathbf{V}$. Questo produce una matrice di attention $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ contenente i valori di self attention di ogni parola di input.

L'uso di matrici in questo contesto facilita il calcolo parallelo degli score, permettendo di processare l'intera sequenza di input in un unico passaggio computazionale.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{2.3}$$

Graficamente questa procedura, chiamata nel paper *scaled dot product attention*, si riassume nello schema in figura 2.17 (lo step di mascheratura opzionale verrà chiarito in seguito).

MultiHead self-attention Il processo di attention viene ulteriormente raffinato introducendo un'architettura *MultiHead*. Questo significa che la stessa sequenza di input viene processata *parallelamente* da più layer, chiamati *head*, di attention. Questo avviene proiettando linearmente le matrici di query $\mathbf{Q}$, chiavi $\mathbf{K}$ e valori $\mathbf{V}$ calcolate in precedenza in h (nel paper originale $h = 8$) spazi diversi, moltiplicandole per le seguenti matrici di pesi

$$\mathbf{W}_i^{\mathbf{Q}}, \mathbf{W}^{\mathbf{K}}{}_i, \mathbf{W}^{\mathbf{V}}{}_i \quad \forall i \in [0, h-1]$$

apprese in fase di addestramento. Una volta ottenute queste matrici proiettate negli h nuovi spazi è possibile calcolare l'attention per ogni head in maniera analoga a quella mostrata nell'equazione 2.3, sostituendo però opportunamente

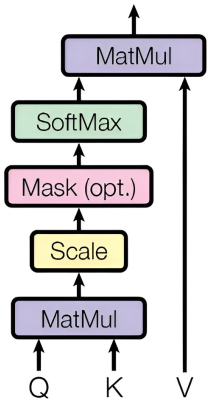


Figura 2.17: Scaled dot product attention schema.

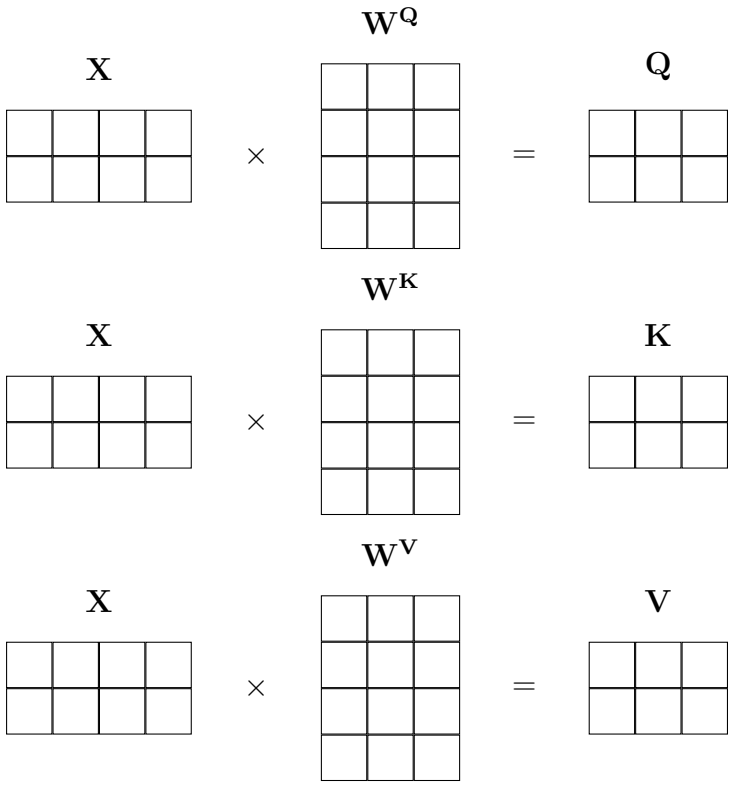


Figura 2.18: Prodotto matriciale per determinazione delle matrici query, key, value.

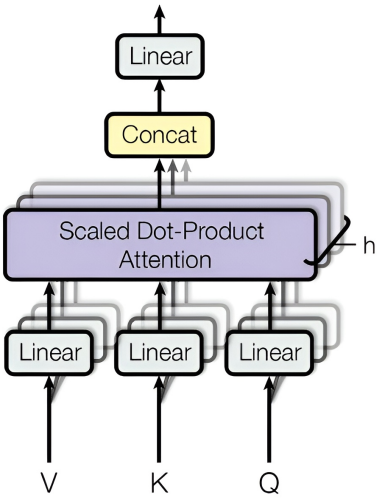


Figura 2.19: MultiHead Attention schema.

le matrici. Indichiamo con head_i la matrice risultante dal calcolo dell'attention sull'head i-esima.

$$\text{head}_i = \text{Attention}(\mathbf{QW}_i^{\mathbf{Q}}, \mathbf{KW}_i^{\mathbf{K}}, \mathbf{VW}_i^{\mathbf{V}})$$

Poiché la rete feedforward posta a valle di ogni enconder si aspetta una sola matrice, occorre trovare un modo di passare loro tutti gli h head prodotti. Pertanto l'ultimo passaggio che resta da fare è concatenare gli h head e farne il prodotto scalare per una matrice di pesi $\mathbf{W}^O$, anch'essa appresa in fase di addestramento. Questo si traduce nella seguente equazione

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)\mathbf{W}^O$$

Lo schema in Figura 2.19 mostra quanto appena descritto a parole. Le tre matrici originali vengono proiettate in h spazi vettoriali tramite delle trasformazioni lineari (prodotto per matrici di pesi), parallelamente su ciascuna di esse viene calcolata una matrice di self attention. Queste matrici vengono concatenate e poi trasformate linearmente mediante il prodotto scalare per una matrice di pesi.

L'implementazione della MultiHead Self-Attention incrementa la capacità del modello di focalizzarsi su differenti posizioni all'interno del testo in input. Una singola matrice di attention, infatti, incorpora elementi di ogni altra parola presente, ma la predominanza potrebbe essere attribuita all'effettiva parola su cui l'attention è calcolata. Utilizzando più head si riesce ad avere una rappresentazione più globale delle relazioni che intercorrono fra ciascuna parola di input con ciascuna altra.

Positional Encoding Poiché l'architettura a transformers non fa uso di ricorrenza (ovvero non è basata su RNN), gli autori del paper [13] hanno anche introdotto alla base dello stack di encoders/decoders, un cosiddetto "*positional encoding*" per permettere al modello di comprendere la posizione relativa o assoluta delle parole all'interno delle sequenze in input. Questo encoding può essere fatto in diversi modi, tuttavia quello scelto dagli autori è il seguente

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$$

dove

- *pos* rappresenta la posizione di un determinato token all'interno della sequenza in input.
- i varia da 0 a d_{model} che rappresenta la dimensione degli embeddings.

Queste funzioni hanno la proprietà di permettere al modello di apprendere facilmente a prestare attenzione alle posizioni relative dei token, indipendentemente dalla lunghezza della sequenza.

Questi positional encoding vengono integrati all'interno della matrice degli embeddings semplicemente sommando elemento per elemento la matrice degli embeddings e i corrispettivi encoding posizionali.

LayerNorm A seguito di ogni layer dei transformer (i.e. Multi-Head Attention, FNN, ecc.) viene applicata un'operazione di normalizzazione, come descritto in [14]. L'obiettivo principale di questa operazione è stabilizzare e accelerare l'addestramento delle reti neurali profonde, facilitando la loro convergenza. L'operazione di normalizzazione può essere rappresentata come:

$$\text{LayerNorm}(\mathbf{X} + \mathbf{Z})$$

dove:

- **X** rappresenta la matrice degli embeddings (a cui sono stati precedentemente sommati i positional encoding).
- **Z** indica la matrice di output prodotta dal layer precedente all'operazione di normalizzazione.

Funzionamento del decoder Tutta la tecnologia alla base della componente del decoder è già stata ampiamente trattata ai paragrafi precedenti. Come mostra Figura 2.15, il decoder si distingue dall'encoder principalmente per due aspetti:

1. Oltre al layer di self-attention e al layer feedforward presenti nell'encoder, il decoder introduce un layer di multihead attention tra questi due. Questo layer consente al decoder di focalizzarsi sull'output dell'encoder, essenziale per operazioni come la traduzione, dove il decoder deve fare riferimento all'input originale (nel contesto di una lingua di partenza) per generare un output appropriato (in una lingua di destinazione).

2. Il layer di self-attention nel decoder è *mascherato* per preservare la sua natura *auto-regressiva*. Questo garantisce che ogni predizione dipenda solo da posizioni precedenti, evitando l'attenzione verso posizioni future. Ciò è realizzato mascherando (impostando a $-\infty$) determinate connessioni, prima di applicare la softmax per quelle che corrispondono a connessioni "illegali" (ossia, connessioni a posizioni future), come illustrato in Figura 2.17 (Mask opt.). Dopo aver applicato la softmax a valori molto negativi, infatti, questi diventano essenzialmente zero, impedendo di fatto l'attenzione verso le posizioni future.

Linear e Softmax Layers Dopo che l'output di tutti i decoder è stato elaborato, abbiamo una serie di vettori numerici. Per convertire questi vettori in parole, vengono applicati i layer *Linear* e *Softmax*.

- Il layer *Linear* è una rete feedforward che proietta i vettori output dallo stack di encoder in uno spazio vettoriale più grande, corrispondente, ad esempio, a un dizionario di 10.000 parole apprese in fase di training. Il vettore in questione è chiamato *vettore di logits*.

- Il layer *Softmax* converte gli score prodotti dal layer Linear in probabilità, selezionando l'output basato sulla cella con la probabilità più alta.

In conclusione l'architettura Transformer complessiva e ad alto livello è rappresentata graficamente in figura 2.20.

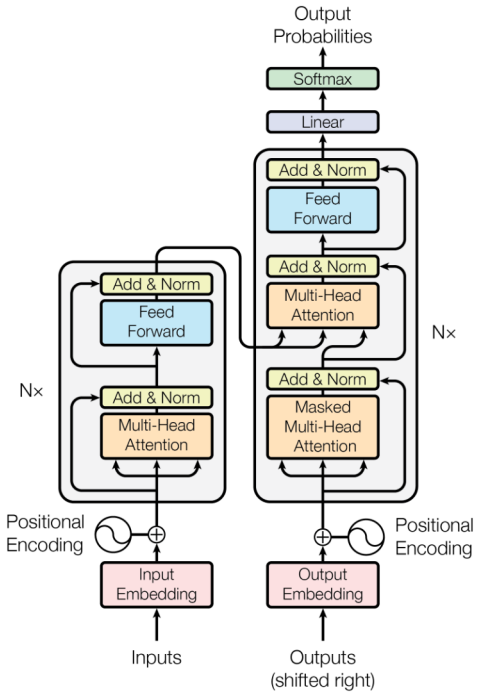


Figura 2.20: Architettura Transformer.

Capitolo 3

SciLay

Questo capitolo tratterà in maniera preliminare le modalità con cui si è deciso di strutturare il dataset *SciLay* creato. Questa panoramica serve da introduzione per contestualizzare il lavoro futuro, mentre i dettagli verranno affrontati nei capitoli successivi.

3.1 Origine e creazione del dataset

Come anticipato nell'introduzione, nel panorama scientifico attuale vi è una forte carenza di articoli che dispongano di lay summary. Probabilmente a causa di questo vuoto, in letteratura è raro trovare dataset preesistenti che includano decine di migliaia di documenti provenienti da differenti riviste. Di conseguenza, è sorta l'esigenza di crearne uno su misura. Opera, questa, di non semplice realizzazione per via di tale carenza, richiedendo un'attenta selezione e cura dei dati disponibili. La prima fase di questo progetto ha implicato un'accurata analisi delle riviste scientifiche che proponessero lay summary, così come elencate sul sito di KTDRR. La ricerca, purtroppo, ha portato a risultati non ottimali rispetto agli obbiettivi preposti. Numerose riviste limitavano l'accesso ai loro articoli mediante sottoscrizioni, mentre quelle che offrivano accessi gratuiti presentavano una disponibilità estremamente limitata, spesso circoscritta a 100-200 articoli per ogni rivista. Tra le fonti più rilevanti, si annoverano Wiley-Journal, Cochrane, PNAS, NIHR e PLoS Medicine. Una problematica addizionale è stata la predominanza del formato PDF tra le pubblicazioni, un formato notoriamente complesso da analizzare[1]. In risposta a tali difficoltà, si è valutata l'opportunità di adottare un diverso approccio metodologico, privilegiando formati più agevolmente analizzabili. Sebbene

[1]Diverse metodologie sono state esplorate, tra cui l'uso di librerie standard come PyPDF2 e l'impiego di soluzioni AI, come Grobid e Science Parser, con quest'ultima che ha fornito i migliori risultati.

l'idea iniziale fosse orientata verso il web scraping delle pagine HTML degli articoli, la mancanza di standardizzazione tra le pagine delle varie riviste ha reso l'approccio non fattibile. Fortunatamente, ulteriori indagini hanno condotto alla scoperta di un database fornito da PubMed, che proponeva articoli in formato XML disponibili per il download massivo. Una volta acquisiti, si è notato che tali documenti presentavano una struttura uniforme, rendendo possibile una raccolta sistematica dei dati rilevanti, riuscendo a filtrare le parti rilevanti per lo scopo del progetto.

3.2 Caratteristiche principali

Dall'analisi di diversi articoli, è stato generato un dataset composto da $43,790$ istanze. Di seguito è descritta nel dettaglio la struttura scelta per il dataset in questione.

3.2.1 Struttura delle istanze

Ciascuna delle $43,790$ istanze presenta la seguente struttura:

- **DOI**. Standard internazionale utilizzato per garantire l'identificazione duratura e univoca di oggetti digitali di varia natura. Il DOI è adottato per identificare ciascuna istanza del dataset prodotto, fornendo un collegamento diretto ai metadati associati all'articolo di riferimento.

- **PMCID**. Identificativo univoco all'interno del database di PMC (PubMed Central, libreria open access di PubMed). L'uso del PMCID facilita l'accesso e la navigazione all'interno di PMC, permettendo di localizzare specifici articoli senza necessariamente fare riferimento al DOI.

- **Plain Summary**. Questo campo corrisponde al lay summary, ovvero un riassunto accessibile e comprensibile dell'articolo in esame.

- **Technical Summary**. Si riferisce all'abstract dell'articolo, fornendo una sintesi tecnica dei contenuti presentati.

- **Full Text**. Rappresenta l'intero contenuto dell'articolo scientifico.

- **Journal**. Indica la rivista scientifica in cui l'articolo è stato pubblicato. Questa informazione fornisce uno sfondo sul contesto e la reputazione della fonte di ricerca. Si noterà che, in questa tesi, il dataset viene analizzato anche in base alle singole riviste di pubblicazione.

- **Topics**. Categorizzazione dell'articolo in base alla sua natura, come ad esempio se è una ricerca, una review o altri tipi di pubblicazioni.

- **Keywords**. Le parole chiave associate all'articolo, utili per definire e contestualizzare l'ambito e il focus del documento.

3.2.2 Suddivisione in subset per rivista

Come anticipato, il dataset è anche stato suddiviso in subset a seconda del giornale di appartenenza. Questa modalità di lavoro presenta una serie di benefici chiave che consentono di condurre esperimenti più attendibili e generalizzabili. In primo luogo questo sistema tiene conto della *specificità delle riviste*: ogni rivista può avere uno stile, un focus o una specializzazione unici. Suddividendo il dataset per rivista, è possibile analizzare e comprendere meglio queste specificità. Un ulteriore vantaggio offerto da questa metodologia è la possibilità di implementare una *validazione incrociata*. Questa tecnica prevede l'addestramento di un modello su uno o più subset, per poi valutare le sue prestazioni su subset differenti. Questo approccio non solo assicura una maggiore robustezza dei risultati ottenuti, ma consente anche di verificare l'efficacia e l'applicabilità del modello in diversi contesti editoriali. Infine, l'adozione di questa strategia di suddivisione facilita l'addestramento di *modelli specifici* per ciascuna rivista. Grazie a tale specializzazione, è possibile ottimizzare le prestazioni del modello rispetto allo stile e alla terminologia propri di una determinata rivista, garantendo risultati di elevata qualità e pertinenza. In sintesi, la segmentazione del dataset in base al giornale di appartenenza si rivela un approccio strategico e metodologico di fondamentale importanza per garantire la validità e la rilevanza delle analisi condotte. Più nel concreto, a seguito dell'analisi delle istanze collezionate, si è deciso di generare dei subset solamente laddove la rivista fornisse un numero di istanze che fosse $\geq 1\%$ rispetto al numero di articoli totali. Questo per evitare di avere split con un numero di istanze irrisorio. Oltre a questo è stata ovviamente messo a disposizione l'intero dataset. La Figura 3.1 mostra la distribuzione percentuale delle istanze per ogni rivista rispetto al totale. La voce "*Others*" corrisponde all'insieme delle riviste con istanze insufficienti per avere un subset autonomo.

La suddivisione ha seguito fedelmente la distribuzione mostrata nella figura in questione, cioè si sono generati 13 subset oltre a quello che comprende le riviste con meno istanze. La tabella 3.1 mostra in termini numerici la dimensione di ogni subset.

Per questioni di sintesi, i nomi delle riviste sono stati abbreviati come segue:

- NC: Nature Communications.

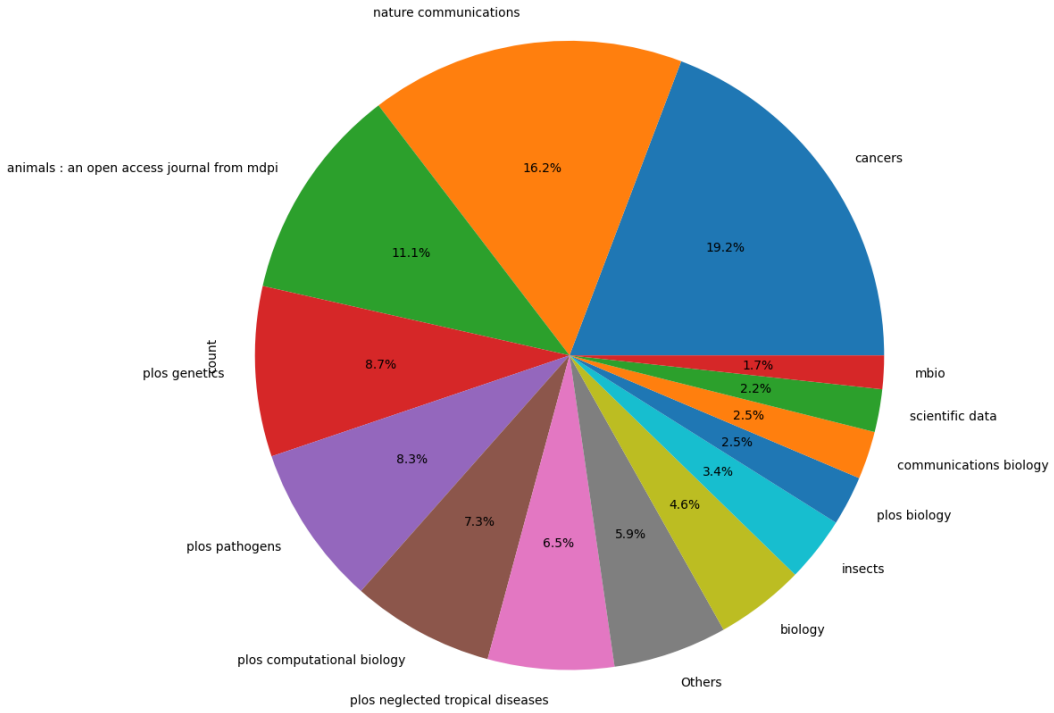


Figura 3.1: Distribuzione percentuale delle istanze per rivista.

	train	validation	test
all	35026	4380	4384
NC	5549	694	694
A	3909	489	489
PLGEN	3087	386	386
PLPAT	2920	365	365
PLCB	2589	324	324
PLNTD	2289	286	287
B	1617	202	203
I	1181	148	148
PLB	896	112	113
CB	867	108	109
SD	725	91	91
MBIO	607	76	76
C	6782	848	848
OTHER	2008	251	251

Tabella 3.1: Dimensione numerica di ogni subset.

- A: Animals : An Open Access Journal from MDPI.
- PLGEN, PLAPAT, PLCB, PLNTD, PLB: PLoS {Genetics, Pathogens, Computational Biology, Neglected Tropical Diseases, Biology}.
- B: Biology.
- I: Insects.
- CB: Communications Biology.
- SD: Scientific Data.
- MBIO: mBio.
- C: Cancers.

3.3 Upload del dataset su HuggingFace

Come ultima operazione effettuata sui dati, si è deciso di caricare l'intero dataset su *HuggingFace* nel repository intitolato, per l'appunto, *sci_lay*. Hugging Face è diventata una delle principali piattaforme per il Natural Language Processing (NLP) e il Deep Learning, offrendo un'ecosistema completo per la formazione, il testing e la distribuzione di modelli di linguaggio. Caricare un dataset su Hugging Face può offrire una serie di vantaggi chiave, in particolare per la comunità di ricercatori e sviluppatori. Ecco una descrizione dei motivi principali:

- **Accessibilità**: Una volta caricato su Hugging Face, il dataset diventa facilmente accessibile a chiunque nella comunità. Questo permette ad altri ricercatori e sviluppatori di sfruttare il dataset senza passare attraverso processi di download e setup complicati.
- **Standardizzazione**: Hugging Face offre una struttura standardizzata per i dataset, rendendo più semplice per gli utenti navigare, comprendere e utilizzare diversi dataset in modo coerente.
- **Integrazione con Modelli**: Hugging Face non è solo una piattaforma per dataset, ma anche per modelli di linguaggio. Caricando un dataset lì, si facilita la formazione, il fine-tuning e la valutazione dei modelli usando quella specifica risorsa, tutto all'interno dello stesso ecosistema.

- **Versioning**: Proprio come per i software, i dataset possono subire modifiche e revisioni nel tempo. Hugging Face supporta il versioning, permettendo agli utenti di accedere a diverse versioni del dataset e tracciare le modifiche. Nel caso di SciLay, ad esempio, il dataset è disponibile solo nella versione 1.0.0 (cased) che dispone degli articoli con caratteri maiuscoli e minuscoli.

- **Documentazione e Metadati**: La piattaforma permette di fornire documentazione dettagliata, basata sul concetto di *Dataset Card* proposta nell'articolo "Data Cards: Purposeful and Transparent Dataset Documentation for Responsible AI"[15], e metadati associati al dataset. In questo modo si aiutano gli utenti a comprendere e utilizzare al meglio la risorsa.

In conclusione, caricare un dataset su Hugging Face non solo semplifica la distribuzione e l'adozione della risorsa, ma promuove anche la standardizzazione, la collaborazione e l'innovazione nell'ambito del Natural Language Processing e delle discipline correlate.

Capitolo 4

Modellazione del progetto

Mentre i capitoli precedenti hanno approfondito l'esplorazione delle tecnologie esistenti e l'analisi dei dati, questo capitolo funge da collegamento cruciale tra la teoria e la pratica, offrendo uno sguardo approfondito sulla modellazione effettiva del progetto. In questo capitolo discuteremo le decisioni di design del nostro sistema, le scelte di modellazione intraprese e come le tecnologie appena introdotte possano essere adattate e ottimizzate per affrontare il problema in questione.

4.1 Infrastruttura e Risorse

Fin dal primo momento ci si è trovati di fronte alla sfida di eseguire esperimenti su modelli con una notevole complessità computazionale. Al giorno d'oggi, infatti, anche i modelli di intelligenza artificiale più basilari richiedono una potenza di elaborazione che solo GPU di alta qualità possono fornire. A questo proposito il professor Moro ha generosamente offerto l'accesso al suo server, garantendomi una capacità di calcolo che non avrei potuto ottenere altrimenti. Il server dispone infatti di sei NVIDIA GeForce GTX 3090 da 24GB di VRAM e altre GeForce GTX TITAN XP da 12GB di VRAM.

Data la posizione fisica remota del server e il fatto che sia condiviso con altri tesisti e ricercatori, si è resa necessaria una gestione accurata delle risorse in un contesto di utilizzo condiviso. Queste considerazioni hanno guidato l'adozione di tecnologie come:

- *Slurm*
- *Docker*
- *SSH*

che verranno trattate nel dettaglio nelle sottosezioni successive.

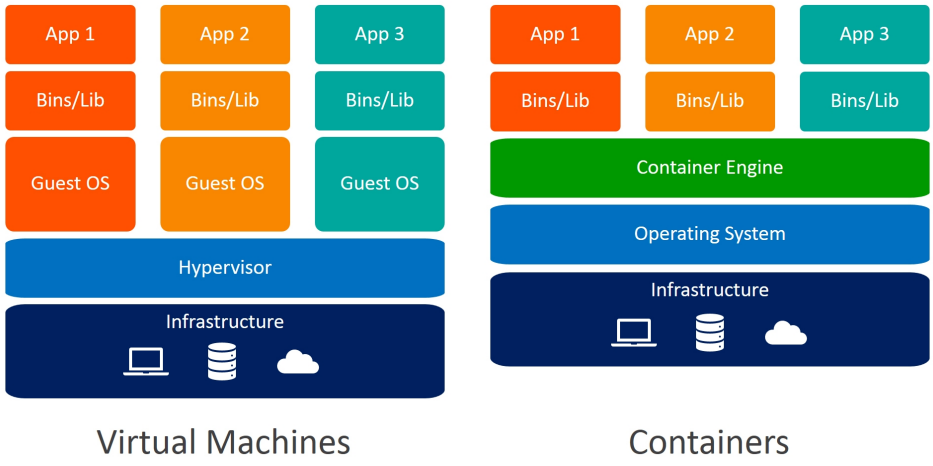


Figura 4.1: Confronto fra stack VM e container.

Fonte: https://images.contentstack.io/v3/assets/blt300387d93dabf50e/bltb6200bc085503718/5e1f209a63d1b6503160c6d5/containers-vs-virtual-machines.jpg

4.1.1 Docker

Docker è una piattaforma software open-source che consente di creare, eseguire e gestire applicazioni all'interno di contenitori (anche detti *container*). Un *container* è una unità standardizzata di software che racchiude il codice e tutte le sue dipendenze in modo che l'applicazione possa essere eseguita in modo uniforme e coerente su qualsiasi ambiente. Questi contenitori sono leggeri, dato che non richiedono l'overhead di una macchina virtuale completa, ma si avvalgono dello stesso kernel del sistema operativo sottostante.

Perché scegliere Docker su una VM

Come anticipato, una delle principali differenze fra un sistema di containerizzazione e una virtual machine (VM) è il fatto che ogni contenitore si appoggia sul sistema operativo della macchina in cui viene eseguito, come mostrato in figura 4.1.

Questa semplice differenza comporta una serie di vantaggi risultati essenziali, fra i principali si riportano:

- *Leggerezza.* I container condividono lo stesso sistema operativo (OS) host, mentre ogni VM ha il proprio sistema operativo completo. Questo significa che i container tendono a utilizzare meno risorse rispetto alle VM.

- *Avvio più veloce.* Poiché i container non hanno bisogno di avviare un intero sistema operativo, possono essere avviati in pochi secondi, mentre le VM potrebbero richiedere minuti.

- *Maggiore densità.* Grazie alla loro leggerezza, è possibile eseguire molti più container su un host rispetto a quante VM potrebbero essere eseguite sulla stessa infrastruttura.

- *Portabilità.* I container sono progettati per essere portatili. Con tecnologie come Docker, è possibile "containerizzare" un'applicazione con tutte le sue dipendenze e eseguirla in modo coerente su diversi ambienti, dallo sviluppo alla produzione.

- *Scalabilità.* La capacità di avviare rapidamente i container li rende ideali per soluzioni scalabili, dove nuove istanze di un'applicazione devono essere avviate o fermate rapidamente in risposta al carico.

Isolamento e Riproducibilità

Le due caratteristiche di fondamentale importanza per lo svolgimento di questo progetto sono l'*isolamento* e la *riproducibilità*. Poiché il cluster condiviso di GPU poneva il problema di mantenere dei blocchi funzionali indipendenti dal nodo in uso, che contenessero tutto il necessario per poter eseguire del codice di deep learning, Docker si è rivelato di vitale importanza. In particolare poiché fornisce

- *Ambienti di lavoro isolati.* Docker consente di creare contenitori che funzionano in modo isolato dal sistema host e dagli altri contenitori, garantendo che le risorse, come la CPU, la memoria e la GPU, assegnate a un contenitore non interferiscano con altri contenitori o con il sistema host. In un ambiente clusterizzato, ciò è essenziale per garantire che ogni job riceva le risorse di cui ha bisogno senza conflitti.

- *Riproducibilità.* Una delle sfide principali nella ricerca scientifica è assicurarsi che gli esperimenti siano riproducibili. Grazie alle immagini Docker, è possibile specificare esattamente quali software, librerie e versioni sono necessarie. Questo significa che, fornendo il container (o l'*immagine* da cui crearlo), l'ambiente di esecuzione rimarrà lo stesso indipendentemente dall'host sul quale viene eseguito, garantendo così la riproducibilità degli esperimenti.

- *Gestione delle dipendenze.* In un contesto di sperimentazione di baseline di abstractive summarization è essenziale scegliere le dipendenze (nonché

le versioni) giuste. Docker consente di semplificare questa gestione. Ad esempio, se il progetto richiedeva una versione specifica di una libreria per la GPU o una particolare versione di una libreria di elaborazione del linguaggio naturale, queste potevano essere specificate nel *Dockerfile*. Questo ha eliminato i problemi derivanti dall'avere versioni diverse delle dipendenze installate su nodi diversi del cluster.

Dockerfile e immagini

Docker si basa su tre concetti fondamentali per implementare il suo approccio alla containerizzazione, come mostrato in figura 4.2. Questi concetti sono:

- Dockerfile
- Immagini
- Containers

Il *Dockerfile* rappresenta una ricetta testuale che Docker utilizza per creare immagini di container. Esso dettaglia una sequenza di istruzioni che Docker segue passo dopo passo. È interessante notare che un singolo Dockerfile può dar vita a diverse immagini.

L'*Immagine* è essenzialmente un pacchetto eseguibile che racchiude tutto ciò che serve per far funzionare un'applicazione: dal codice, alle librerie, alle variabili di ambiente, fino ai file di configurazione.

Il *Container*, d'altra parte, è una versione eseguibile di un'immagine. Quando un'immagine viene avviata, essa prende vita sotto forma di container. Questi container funzionano in maniera isolata, sia tra loro sia rispetto al sistema host, garantendo che ogni container abbia le proprie risorse, come filesystem, connessione di rete e identificatori di processo (PID).

Il flusso di lavoro (anch'esso sintetizzato in Figura 4.2) tipico con Docker include la scrittura di un Dockerfile, la costruzione di un'immagine da quel Dockerfile e, infine, l'esecuzione di un container basato sull'immagine costruita.

Come anticipato, durante la fase di modellazione del progetto, l'adozione di Docker è stata decisiva per assicurare coerenza e riproducibilità nell'ambiente di esecuzione. Visto che il dataset *SciLay* è stato pubblicato su HuggingFace, per garantire consistenza, abbiamo optato per l'utilizzo delle API fornite da HuggingFace attraverso la libreria `Transformers`. Di conseguenza, nella creazione del Dockerfile, ci siamo appoggiati all'immagine base "huggingface/transformers-pytorch-latest-gpu", che fornisce le librerie essenziali di PyTorch e Hugging Face Transformers per GPU, garantendo così la massima efficienza nell'addestramento e nell'inferenza dei modelli.

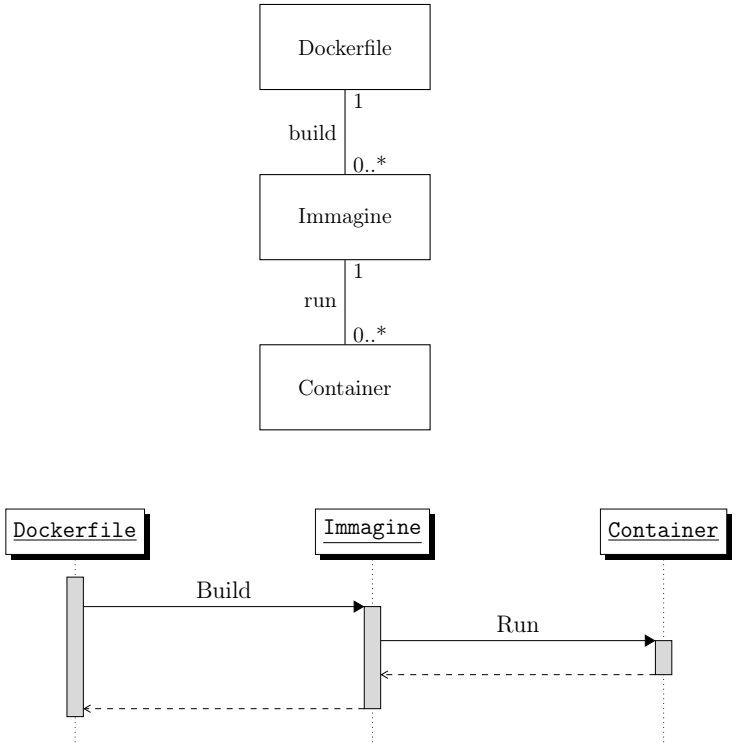


Figura 4.2: Diagrammi delle classi (in alto) e di sequenza (in basso) mostranti le relazioni statiche e dinamiche tra Dockerfile, Immagine e Container.

Oltre a questo, volendo provare i *Large Language Models (LLM)* più recenti come *LLama2* e *Mistral*, si è posta la necessità di avere un container fortemente ottimizzato per le GPU Nvidia a nostra disposizione. Per questo motivo si è deciso di partire da immagini fornite da Nvidia stessa come "nvcr.io/nvidia/pytorch:22.12-py3". Questo fornisce accesso alle libreria CUDA 11.8, le quali rimangono retrocompatibili con le GPU a nostra disposizione (limitate a CUDA 11.6) pur dando la possibilità di usare l'ultima versione di `PyTorch`.

4.1.2 Slurm

Slurm è un sistema open source per la gestione di cluster e la pianificazione di lavori, ideale sia per grandi che piccoli cluster Linux. È resistente ai guasti, altamente scalabile e non richiede modifiche al kernel per funzionare. Slurm svolge tre funzioni principali come gestore di carichi di lavoro su cluster:

1. Assegna l'accesso esclusivo o condiviso alle risorse (nodi di calcolo) agli utenti per un certo periodo di tempo, permettendo loro di eseguire

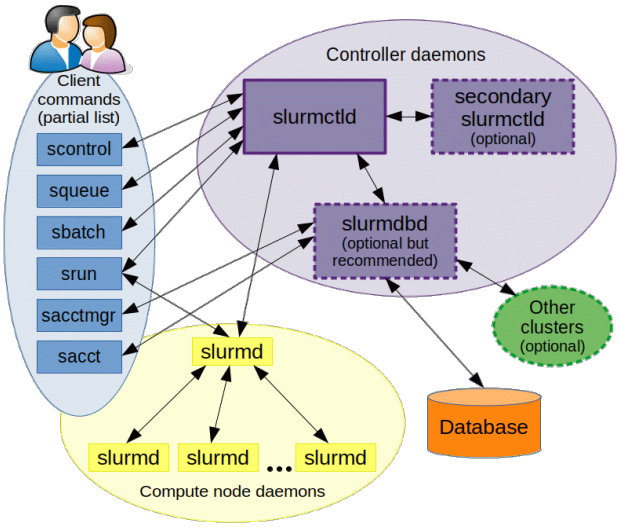


Figura 4.3: Architettura di Slurm.
Fonte: https://slurm.schedmd.com/overview.html

operazioni.

2. Offre una struttura per avviare, eseguire e monitorare lavori, solitamente paralleli, sui nodi assegnati.

3. Gestisce una coda di lavori in attesa, arbitrando l'accesso alle risorse.

Architettura

Come mostrato in Figura 4.3, Slurm utilizza un gestore centralizzato, chiamato **slurmctld**, per monitorare risorse e lavori. Può anche esserci un gestore di backup pronto a sostituirlo in caso di guasto. Ogni server di calcolo (o nodo) ha un demone chiamato **slurmd**, che funziona un po' come una shell remota: attende istruzioni, le esegue, restituisce lo stato e attende altre istruzioni.

Esiste un demone opzionale, **slurmdbd**, che registra informazioni contabili (i.e. monitoraggio delle risorse) per più cluster gestiti da Slurm in un unico database. Esistono anche altri componenti che però verranno trascurati per ragioni di semplicità.

Ogni client può agire sia sul controller centrale (**slurmctld**) che sui CLI associate a ciascun nodo (**slurmd**), lanciando diversi comandi, fra cui

- *srun*: per avviare jobs in modalità interattiva;
- *sbatch*: per avviare jobs in batch;
- *scancel*: per terminare lavori in coda o in esecuzione;

- *sinfo*: per riportare lo stato del sistema;

Nello specifico, nel cluster di GPU utilizzato, all'atto dell'inserimento di un task in coda, viene calcolato un valore intero per definirne la priorità. Questo valore tiene conto di diversi fattori, tra cui l'ordine di inserimento e il bilanciamento di carico tra gli utenti.

Parallelizzazione e Gestione delle Risorse

Nonostante la capacità di Slurm di gestire complessi cluster multi-nodo, si è optato per un approccio semplice ma efficace, utilizzando sempre un singolo nodo per ogni esperimento. Questo ha permesso di avere un controllo totale sulla GPU desiderata e di garantire che ogni esperimento avesse accesso esclusivo alle risorse necessarie.

Nella fase di configurazione dei task, si è preferito adottare un approccio ibrido tra interazione diretta con l'utente ed esecuzione autonoma (in batch) degli esperimenti. A tal proposito, Slurm mette a disposizione due comandi fondamentali: *srun* e *sbatch*. Mentre *sbatch* si rivela la scelta ideale per esperimenti di lunga durata che non necessitano di interventi manuali, nelle fasi iniziali del progetto *srun* risulta essere estremamente utile. Il valore aggiunto di *srun* è la sua capacità di avviare sessioni in modalità interattiva. Questo permette di lanciare un lavoro e, contemporaneamente, di accedere a una shell direttamente nel nodo assegnato. Tale funzionalità si è rivelata particolarmente preziosa per operazioni di debugging, per condurre test in maniera agile o per analizzare i dati in tempo reale. Una volta che il codice è stato ottimizzato e testato, per gli addestramenti prolungati e stabili, si è fatto affidamento su *sbatch*.

Questo approccio, pur essendo semplice, si è dimostrato efficace per le esigenze di questo progetto. Si è potuto sfruttare la potenza del cluster di GPU mantenendo al contempo un elevato livello di controllo e prevedibilità. La combinazione di Slurm con il metodo di utilizzo appena descritto ha permesso di eseguire gli esperimenti in modo efficiente, garantendo risultati tempestivi e affidabili.

4.1.3 Flusso di lavoro complessivo

Complessivamente, come mostrato in Figura 4.4, il flusso di lavoro prevede una fase di connessione al server via SSH (Secure SHell) mediante credenziali fornite dall'amministratore del cluster, Giacomo Frisoni. Una volta connessi e configurata una shell byobu (il cui vantaggio è quello di garantire *shell persistenti*, cioè che rimangano attive anche dopo la chiusura del CLI), data la condivisione delle risorse, si procede inserendosi in una coda di attesa gestita

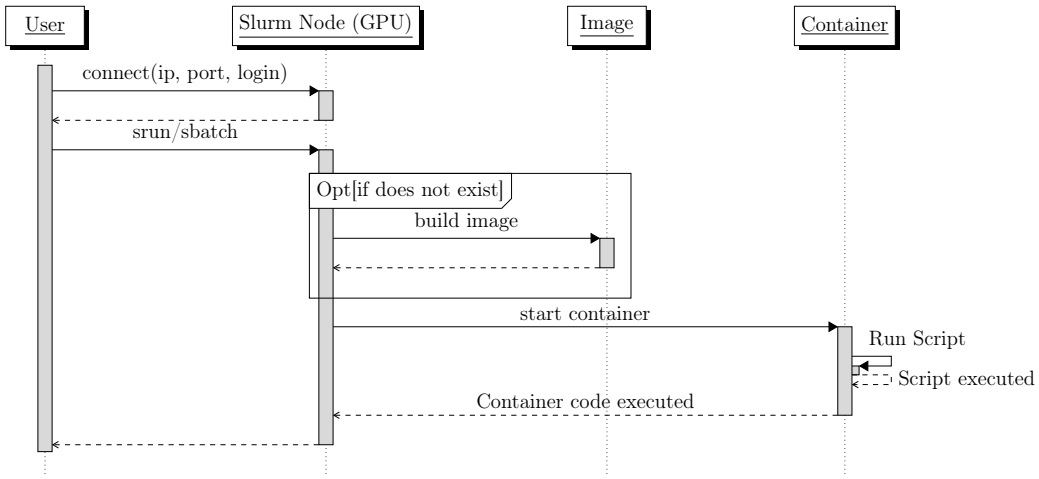


Figura 4.4: Diagramma di sequenza per la gestione del cluster di GPU.

da Slurm. Una volta ottenute le risorse necessarie (cioè è stato allocato un nodo), si porcede con l'avvio degli esperimenti all'interno di specifici container grazie a Docker. Per fare questo, prima è necessario aver creato un'immagine a partire dal Dockerfile, dunque formalmente avviene il controllo mostrato in figura. All'interno del container, una volta istanziato, si esegue uno script sh che lancia un programma python passandogli dei parametri che andranno definiti in fase di sviluppo del progetto.

Considerazioni pratiche

Una complicazione è emersa dal fatto che Slurm percepisce ogni nodo come se fosse un componente della stessa unità computazionale. Di conseguenza, richiedendo risorse da un nodo, potrebbero venire allocate su un altro. Per garantire che gli esperimenti vengano eseguiti correttamente, è fondamentale che il codice su ogni macchina ri1manga sincronizzato. Sebbene esistano strumenti dedicati alla sincronizzazione dei file, come Unison, una soluzione efficace è mantenere tutto all'interno di un repository Git, metodo adottato per questo progetto.

4.2 Architettura del sistema

Le recenti innovazioni nel campo del deep learning, in particolare nell'ambito dell'NLP, hanno portato alla creazione di modelli di summarization *astrattivi* che superano ampiamente le performance delle tecniche *estrattive* precedentemente adottate. Da questa constatazione, l'attenzione si è spostata verso questo tipo

di approcci, capaci di interpretare in modo più profondo il contenuto dei testi e di generare sintesi più accurate, evitando le carenze tipiche dei modelli estrattivi come la frammentazione e la mancanza di fluidità nel riassunto.

Considerando la vastità di tecniche astrattive, come discusso nel capitolo 2, è emerso che i modelli basati sull'architettura encoder-decoder RNN presentano delle limitazioni. Una delle loro principali debolezze risiede nella gestione di testi lunghi: di fronte a testi estesi, tendono a troncare il contenuto, limitando la capacità di generare riassunti completi. Questa limitazione è particolarmente critica quando si tratta di documenti come quelli di SciLay, che sono di notevole lunghezza e contengono informazioni rilevanti sparse in tutto il testo.

In questo contesto, l'architettura Transformer si è distinta come soluzione promettente. Questi modelli non solo superano i modelli RNN in termini di accuratezza, ma hanno anche dimostrato di essere particolarmente efficaci nell'analisi di testi lunghi, grazie alla loro capacità di comprendere relazioni a distanza nel testo. Di conseguenza, la scelta è stata quella di basare il nostro sistema esclusivamente su questa architettura.

4.3 Scelte di modellazione

Per i nostri esperimenti sono state utilizzate delle *baseline astrattive*. Questo significa che sono stati selezionati i modelli più promettenti disponibili in letteratura.. La descrizione dei modelli è stata categorizzata in base alla loro architettura sottostante ma sono tutti accomunati dall'utilizzo di transformers piuttosto che RNN.

4.3.1 Modelli Seq2Seq con Attention

I modelli Seq2Seq con Attention sono stati ampiamente descritti nel capitolo 2. Tra questi, i più noti sono

- BART
- T5
- PEGASUS

tutti basati sull'architettura a Transformers.

Bart

BART (Bidirectional and Auto-Regressive Transformer) [7] di Facebook AI è stato progettato per compiti di elaborazione del linguaggio naturale

	BART Base	BART Large
Layer encoder/decoder	6	12
Hidden Size	768	1024
Heads di Attention	12	16
FNN Size	3072	4096
Parametri	140M	340M

Tabella 4.1: Confronto tra Bart Base e Large.

(NLP), come il riassunto e la correzione grammaticale. Il suo metodo di pre-addestramento, *denoising autoencoder*, altera il testo in ingresso in vari modi e poi tenta di ricostruirlo attraverso un approccio *auto-regressivo*. BART è disponibile in diverse varianti, tra cui *base* e *large*, come riassunto nella Tabella 4.1. Sebbene la versione large sia intrinsecamente più potente, richiede anche risorse computazionali significativamente maggiori rispetto alla versione base.

LSG Bart

La versione estesa di BART, denominata LSG BART, incorpora una tecnica avanzata di attenzione chiamata *Local Sparse and Global (LSG) attention*. Questa evoluzione dell'attenzione, come delineato nel paper [16], mira a ridurre la complessità computazionale associata al meccanismo di self-attention tradizionale, pari a $O(n^2)$ rispetto alla lunghezza dell'input, in quanto per ogni token, è necessario considerare ogni altro token nella sequenza. LSG attention è una risposta a questa sfida, proponendo una struttura tridimensionale:

1. *Local Attention.* Si assume che localmente a un token si debbano catturare informazioni più dettagliate sul suo vicinato in modo da catturare le relazioni che intercorrono, appunto, localmente. Questo richiede un'attention più densa, chiamata locale.

2. *Sparse Attention.* Con l'ampliarsi del contesto, l'informazione richiesta diventa più astratta, e un'attenzione densa su tutti i token diventa computazionalmente proibitiva. Sarà perciò sufficiente catturare un'informazione più ad alto livello. Questo rende però necessario sviluppare un modo per espandere il contesto locale con un ulteriore insieme di token selezionati con delle specifiche regole. Capire che token usare è parte dell'attention sparsa.

3. *Global Attention.* L'attenzione globale serve come ponte tra le informazioni catturate dall'attenzione locale e sparse. A differenza dei token ordinari, i token globali possono prestare attenzione a tutti i token nella

	T5-Small	T5-Base	T5-Large	T5-3B	T5-11B
Layer encoder/decoder	6	12	24	24	24
Hidden Size	512	768	1024	1024	1024
Heads di Attention	8	12	16	32	128
Parametri	60M	220M	770M	3B	11B

Tabella 4.2: Confronto tra diverse varianti di T5.

sequenza e viceversa, migliorando il flusso di informazioni all'interno del modello. Pertanto questa attention garantisce una comprensione efficace dell'intera sequenza, facilitando l'elaborazione di input lunghi e rendendo il modello più adatto per compiti che richiedono una comprensione di contesti ampi.

Il paper offre anche un modo per trasformare modelli già pre addestrati nella loro versione LSG. LSG Bart è pertanto una versione di Bart la cui attenzione è stata modificata con l'architettura appena definita.

T5

T5 (Text-to-Text Transfer Transformer) [8] di Google Research è stato ideato per trattare qualsiasi compito NLP come una traduzione di testo in testo. Durante la fase di pre-addestramento, il modello apprende a tradurre frasi con token mascherati nella loro versione completa, addestrandosi in un'ottica di riempimento dei token mancanti. Durante la fase di addestramento su compiti specifici, qualsiasi compito viene formulato come una traduzione, in cui l'input viene tradotto in un output desiderato (ad esempio, domanda-risposta, riassunto, traduzione, ecc.). T5 è disponibile in diverse dimensioni, dalle versioni più piccole alle versioni estremamente grandi come T5-11B. La versione *base* e *large* sono le più comunemente utilizzate nella ricerca e nelle applicazioni pratiche. Le specifiche tecniche delle diverse versioni di T5 variano, ma in Tabella 4.2 vengono riportate le caratteristiche delle principali, per una comparazione diretta.

Pegasus

PEGASUS (Pre-training with Extracted Gap-sentences for Abstractive Summarization) [17] di Google Research, pur essendo ideato principalmente per il riassunto abstrattivo, ha dimostrato versatilità anche in altri compiti NLP. La distintiva strategia di pre-addestramento di PEGASUS consiste nell'identificare e rimuovere alcune frasi dall'input e poi tentare di generare tali frasi omesse. Questo approccio è noto come *gap sentence generation* e ha l'obiettivo di

indurre il modello a imparare una rappresentazione significativa del testo che può essere utilizzata per il riassunto.

LSG Pegasus

Analogamente a LSG Bart, anche Pegasus ha una sua versione LSG che si avvale dei tre tipi di attention precedentemente citati.

4.3.2 Large Language Models

La particolarità di questi modelli è che sono addestrati su miliardi di parametri. Si va da modelli più piccoli basati su 7B di parametri (i.e. Mistral), passando poi per quelli intermedi come LLama2 da 13B per poi arrivare ai modelli più grandi che arrivano a 1.76T di parametri come GPT4. Sebbene sia possibile addestrare ulteriormente i modelli affinché possano essere ulteriormente affinati per performare bene su task specifici (i.e. generazione di riassunti), ai fini di questo progetto ci si è concentrati su esperimenti *zero-shot*. Cioè non si ri addestrano i modelli per questioni di tempo.

Llama2

LLama2 [18] è un insieme di modelli sviluppati da Meta, la compagnia madre di Facebook, come risposta ai modelli GPT di OpenAI e ai modelli IA di Google come PaLM 2. La differenza chiave è che LLama2 è un modello di linguaggio di grandi dimensioni open source, disponibile gratuitamente per scopi di ricerca e commerciali.

Questi modelli sono forniti già pre-addestrati, permettendo così esperimenti zero-shot, con una gamma di dimensioni che va da 7B a 70B di parametri. Esiste una varietà di modelli adattati a compiti specifici. In particolare, per i compiti legati al dialogo o, più ampiamente, alla generazione di linguaggio naturale, Meta ha sviluppato una variante ottimizzata denominata *LLama 2-Chat*, che è la versione che sarà impiegata in questo progetto. Esiste poi una versione *LLama 2-Code* addestrata per compiti relativi allo sviluppo di codice.

Mistral

Modello introdotto il 10 ottobre 2023, appositamente sviluppato per ottenere risultati molto elevati mantenendosi relativamente snello ed efficiente. Come riportato dal paper [19], esso ha infatti superato (o pareggiato) i risultati di LLama 2 13B su tutti i benchmark considerati, pur essendo addestrato su 7B di parametri. Si veda a questo proposito figura 4.5.

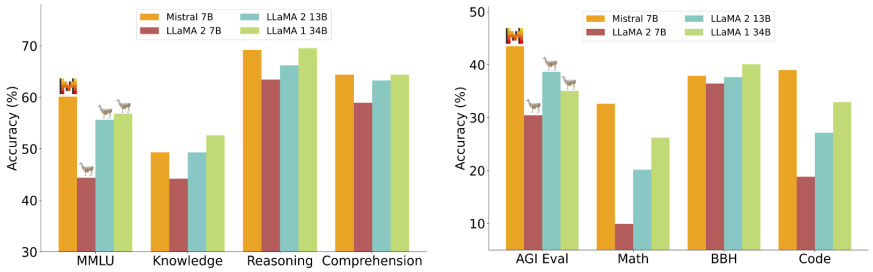


Figura 4.5: Confronto fra Mistral e LLama2 su vari task.

Come per LLama2, anche Mistral offre i suoi modelli già pre-addestrati, per compiti specifici. In particolare *Mistral 7B Instruct* è appositamente ideato per gestire compiti di conversazione.

4.4 Metriche di valutazione

Di seguito vengono riportate le metriche che verranno utilizzate per valutare le performance delle baseline provate. Molti di questi sono diventati uno standard in letteratura per attribuire un punteggio ai modelli di summarization.

4.4.1 Rouge

Introdotto nel paper [20], l'acronimo *ROUGE* sta per *Recall-Oriented Understudy for Gisting Evaluation* ed è diventato uno degli strumenti più utilizzati per valutare la qualità dei riassunti generati dai modelli, comparandoli anche con quelli creati umanamente.

La ricerca originale propone diverse varianti della metrica ROUGE.

- *ROUGE-N*: Misura la sovrapposizione di n-grammi tra il testo generato e il testo di riferimento. La formula seguente

$$\text{ROUGE-N} = \frac{\sum_{S \in \text{Ref}} \sum_{\text{gram}_n \in S} \text{Count}_{Gen}(\text{gram}_n)}{\sum_{S \in \text{Ref}} \sum_{\text{gram}_n \in S} \text{Count}_{Ref}(\text{gram}_n)}$$

  calcola la somma di tutti gli n-grammi nei riassunti generati e la normalizza dividendola per il numero di n-grammi nei riassunti di riferimento. In questo caso gli n-grammi potranno essere di qualunque dimensione. Se $n = 1$ si parla di ROUGE-1, per $n = 2$ ROUGE-2 e cosi via.

- *ROUGE-L*: Calcola la Lunghezza della Sottosequenza Comune (LCS) tra il testo generato e quello di riferimento.

$$\text{ROUGE-L} = \frac{\sum_{S \in \text{Ref}} LCS(\text{Gen}, S)}{\sum_{S \in \text{Ref}} |S|}$$

  La somma delle LCS è normalizzata dividendola per la somma delle lunghezze dei riassunti di riferimento, fornendo una misura della somiglianza fra i testi generati e quelli di riferimento.

- *ROUGE-W*: Misura la sovrapposizione di parole basata sulla LCS pesata tra il testo generato e il testo di riferimento.

$$\text{ROUGE-W} = \frac{\sum_{S \in \text{Ref}} WLCS(\text{Gen}, S)}{\sum_{S \in \text{Ref}} |S|}$$

  dove $WLCS(\text{Gen}, S)$ non è altro che la lunghezza della sottosequenza comune pesata tra il sistema e un riassunto di riferimento.

4.4.2 Bleu

Metrica introdotta nel paper [21] e sta per Bilingual Evaluation Understudy. Anch'essa è ampiamente utilizzata nel contesto delle pubblicazioni relative a task di NLP. In particolare è nata per valutare la qualità delle traduzioni automatiche rispetto a una o più traduzioni di riferimento fornite da umani. Confronta le corrispondenze n-gram tra il testo tradotto e il testo di riferimento, fornendo un punteggio che indica quanto il testo tradotto sia vicino al testo di riferimento.

Nel contesto di questo progetto si sono utilizzate BLEU-1, BLEU-2, BLEU-3 e BLEU-4 che considerano rispettivamente n-grammi di dimensione 1, 2, 3 e 4.

4.4.3 R

Intrdotta nell'articolo [22], è una metrica utilizzata per avere un valore numerico riassuntivo delle varie ROUGE. Nello specifico rappresenta una media ponderata delle metriche ROUGE, dove la varianza delle metriche ROUGE viene utilizzata come fattore di ponderazione per dare maggiore peso alle metriche con minore varianza. Infatti se una metrica ha una varianza inferiore, significa che è più stabile o consistente attraverso differenti esempi. Dare maggiore importanza alle metriche con minore varianza potrebbe contribuire a ottenere una valutazione più affidabile e meno influenzata da outlier o fluttuazioni casuali nei dati. Di seguito è riportata la formula utilizzata per calcolarla:

$$\mathcal{R} = \frac{\mathcal{A}(r_1, r_2, r_L)}{1 + \sigma_r^2}$$

dove, dato un set di valori $(x_1, x_2, \ldots, x_n)$ media e varianza sono definite come segue

$$\mathcal{A}(x_1, x_2, \ldots, x_n) = \frac{1}{n}\sum_{i=1}^{n} x_i$$
$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mathcal{A}(x_1, x_2, \ldots, x_n))^2$$

Con questa definizione $\mathcal{R} \in [0, 1]$. Nei risultati mostrati nel capitolo 5, per normalizzare questa metrica alle altre (cioè avere lo stesso ordine di grandezza), le metriche r_1, r_2, r_L al denominatore vengono divise per 100.

4.4.4 Bert Score

Metrica importante per poter ottenere un valore numerico che esprima la vicinanza semantica del riassunto generato e quello di riferimento.

Come suggerisce il nome, si basa sul modello pre addestrato Bert Score. In particolare, come descritto all'interno del paper [23], BERTScore, in maniera analoga a qualunque altra metrica, produce degli score di similarità numerici fra ogni token generato e ogni token target.

Tuttavia, piuttosto che valutare dei match esatti, questa similarità è calcolata semanticamente a partire dagli embeddings di Bert. Questo fa si che la metrica sia più simile a una valutazione umana poiché coglie la similarità fra i significati dei due testi.

4.4.5 Bart Score

Metrica proposta nel paper [24] per valutare la qualità del testo generato in diverse applicazioni di NLP. La sfida principale in queste applicazioni è valutare se i testi generati siano fluenti, accurati o efficaci. BARTScore affronta questa sfida considerando la valutazione del testo generato come un problema di generazione di testo, modellato utilizzando modelli pre-addestrati di tipo sequence-to-sequence.

Come suggerisce il nome, la metrica utilizza il modello BART per convertire il testo generato in un testo di riferimento o nel testo sorgente, e valuta la qualità del testo generato in base al successo di queste conversioni. In altre parole, se il modello BART riesce a convertire con successo il testo generato nel testo di riferimento o nel testo sorgente, il testo generato viene considerato di alta qualità, e riceverà un punteggio BARTScore più alto.

L'idea generale dietro questo approccio è che i modelli addestrati a convertire il testo generato in un testo di riferimento o nel testo sorgente otterranno punteggi più alti quando il testo generato è di migliore qualità.

4.4.6 Mean Cosine Similarity

La *Similarità Coseno* è una misura di similarità tra due vettori in uno spazio multidimensionale. Può essere usata nel contesto dell'NLP per comparare la similarità tra documenti rappresentati come vettori. Come descritto nel Capitolo 2 è possibile convertire ogni parola in un vettore che preservi la sua semantica. Questo consente di poter fare operazioni vettoriali che si riflettano sulle parole stesse.

Dati due generici vettori $\mathbf{x}, \mathbf{y}$, la loro similarità si calcola sommando i prodotti scalari dei vettori (sommando i prodotti di coppie di componenti) e dividendo questa quantità per le rispettive norme, cioè

$$\text{Cosine Similarity} = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||||\mathbf{y}||} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}} \tag{4.1}$$

Supponiamo ora di avere due rappresentazioni vettoriali di documenti A, B composti da n parole A_i, B_i per $i \in [1, n]$ (anch'essi vettori). La similarità media dei due documenti è data dalla media della similarità coseno 4.1, per ogni coppia di parole.

$$\text{Mean Cosine Similarity} = \frac{1}{n}\sum_{i=1}^{n} \frac{A_i \cdot B_i}{||A_i||||B_i||}$$

I benefici di lavorare sugli angoli come fa, appunto, la *similarità coseno* è quella di non essere influenzati dalla lunghezza dei vettori. Questo risulta molto utile quando occorre calcolare la similarità fra documenti di dimensioni notevolmente diverse. Dunque questa metrica è utile per normalizzare, cioè valutare la proporzione delle occorrenze fra i due testi, piuttosto che la dimensione effettiva.

4.4.7 Mean Generation Length

La *Mean Generation Length* (MGL) è una metrica utilizzata per valutare la lunghezza media delle sequenze o dei testi generati da un modello di generazione del linguaggio naturale. Questa metrica può fornire insight utili sull'abilità di un modello di generare risposte concise o, al contrario, su quanto tende a generare risposte prolisse.

Dato un insieme di n documenti generati, denotati come prediction_i per $i \in [1, n]$, la MGL è calcolata come segue:

$$\text{MGL} = \frac{1}{n} \sum_{i=1}^{n} \text{len}(\text{prediction}_i)$$

4.4.8 Carburacy

Altra metrica introdotta nel paper [22], il cui scopo è attribuire un punteggio al modello basandosi non solo sulle sue prestazioni effettive, ma anche sull'impatto ambientale che il suo addestramento richiede. Nello specifico, *carburacy* indicato con Υ, dipenderà dal punteggio $\mathcal{R}$ ottenuto (direttamente), e dal costo $\mathcal{C}$ richiesto dal modello per ottenere quel punteggio (inversamente). La formula è la seguente

$$\Upsilon = \frac{e^{\log_\alpha \mathcal{R}}}{1 + \mathcal{C} \cdot \beta}$$

in cui $\alpha \geq 0, \beta \geq 0$ sono iperparametri necessari a bilanciare rispettivamente le prestazioni e il costo.

Capitolo 5

Sviluppo

Si procede con la descrizione del lavoro svolto sulle fonti per la costruzione del dataset, e lo sviluppo del codice per poter usare le baseline descritte nei capitoli precedenti.

5.1 Creazione del dataset

Come anticipato, dopo aver individuato le fonti per l'estrazione dei documenti, è stato sviluppato il codice necessario per poterli raccogliere in un formato semi-strutturato. A questo scopo, si è optato per l'aggregazione dei dati in file `jsonlines`. I vantaggi di questo formato rispetto ad altri, come ad esempio `CSV`, includono i seguenti:

- *Facilità di elaborazione incrementale.* `Jsonlines` gestisce un singolo oggetto `JSON` per riga, rendendo molto più semplice l'elaborazione dei dati in modo incrementale senza dover caricare l'intero set di dati in memoria. Questo lo rende ideale per l'elaborazione di flussi di dati o file di grandi dimensioni.

- *Facilità di debug.* A causa di errori nella costruzione del codice o di strutture non consistenti nell'XML, si è reso necessario debuggare i file prodotti. In caso di errori nei dati, è molto più semplice individuare e correggere un errore in una specifica riga di un file `jsonlines` rispetto a cercarlo all'interno di un grande blocco di `JSON`.

- *Leggibilità.* Mentre un file `CSV` definisce il ruolo di ciascuna colonna solo nella prima riga e separa i campi con punti e virgola, spazi, ecc., il formato `jsonlines` scelto, come illustrato nel Listato 5.1, permette di identificare immediatamente tutti i campi di ogni istanza con una rapida occhiata.

```
{
    "doi": "10.1038/s41467-020-16904-3",
    "pmcid": "PMC7308400",
    "plain_text": "Single cell RNA-seq is a powerful method to...",
    "technical_text": "Single-cell RNA sequencing (scRNA-seq) is...",
    "full_text": "Introduction Tissues are complex milieus comprising...",
    "journal": "nature communications",
    "topics": [ "Article" ],
    "keywords": [ "RNA sequencing", "Computational models", "Software" ]
}
```

Listato 5.1: Formato jsonlines articoli.

```
<sec title="Simple Summary">
  <!-- Contenuto del summary -->
</sec>

<abstract abstract-type="summary">
  <!-- Contenuto del summary -->
</abstract>

<abstract abstract-type="executive-summary">
  <!-- Contenuto del summary -->
</abstract>

<abstract id="Abs">
  <!-- Contenuto del summary -->
</abstract>
```

Listato 5.2: Struttura XML di un summary.

Poiché non tutti i documenti scaricati presentavano la struttura desiderata (dunque integrassero un summary), si è dovuto fare parsing di questi alla ricerca di tale campo. Posto che alcuni documenti possedessero tale riassunto, non tutti utilizzavano lo stesso tag per identificarlo all'interno del *DOM* (Document Object Model). Per questo motivo si è reso necessario un approccio ad'hoc per ogni tipologia di tag identificato a seguito di un'ispezione manuale.

Di seguito sono riportate le strutture più frequentemente utilizzate per definire tali riassunti.

Per fare parsing dei documenti `XML` si è deciso di utilizzare il modulo `xml.etree.ElementTree` di Python che offre un'API semplice da usare e molto efficiente.

In particolare è stato utilizzato script 5.3 per estrarre una stringa in formato json per ciascun documento il cui summary è nel formato `<sec title=''Simple Summary''>`.

Il codice presentato è molto semplificato e non tiene conto di tutti gli altri campi estratti, per i quali occorrerebbe fare un ragionamento analogo a quello

```python
import xml.etree.ElementTree as ET

def retrieve_simple_summary(xml_file):
    tree = ET.parse(xml_file)
    root = tree.getroot()

    summary_section = root.find(".//sec[title='Simple Summary']")
    summary_content = ""
    if summary_section is not None:
        summary_content = ET.tostring(summary_section, encoding='unicode', method='text')
        summary_content = " ".join(summary_content.split()).strip()

    return summary_content
```

Listato 5.3: Script Python per parsing della sezione 'Simple Summary'.

```python
import xml.etree.ElementTree as ET

def retrieve_summary_abstract(xml_file):
    tree = ET.parse(xml_file)
    root = tree.getroot()

    summary_abstract = root.find(".//abstract[@abstract-type='summary']")
    summary_content = ""
    if summary_abstract is not None:
        summary_content = ET.tostring(summary_abstract, encoding='unicode', method='text')
        summary_content = " ".join(summary_content.split()).strip()

    return summary_content
```

Listato 5.4: Python script per estrarre l'abstract di tipo 'summary'.

dei summary, in quanto anch'essi non aventi struttura consistente su tutti i documenti.

Si è proceduto in maniera analoga per la tipologia di summary nel formato `<abstract abstract-type=''summary''>` con il codice listato a 5.4.

Il processo è stato ripetuto in maniera similare per tutte le altre tipologie di XML, non vengono riportate per ragioni tipografiche.

5.2 Preprocessamento

Dopo aver raccolto tutte le istanze di interesse, si è proceduto con l'analisi della qualità del dataset appena creato.

In particolare, si è caricato il file in formato `jsonlines` all'interno di un `DataFrame Pandas`. Questa decisione è stata presa basandosi sull'ampio ventaglio di benefici offerti dalla libreria Pandas nella gestione dei dati. Le

strutture dati di Pandas, come i DataFrame, sono ottimizzate per l'elaborazione ad alte prestazioni di grandi volumi di dati tabulari.

5.2.1 Analisi delle istanze mancanti

Si è utilizzato il metodo `df.describe()` per ottenere una panoramica immediata della struttura e distribuzione dei dati. Questo passaggio è cruciale per comprendere la natura del dataset e pianificare eventuali operazioni di preprocessing.

La Tabella 5.1 presenta il risultato dell'applicazione di `df.describe()` al nostro dataset.

	Count	Unique	Top	Freq
doi	46486	44142		2345
pmcid	46486	44143		2344
plain_text	46486	46470	We are planning to update...	2
technical_text	46486	46478	Chronic myeloid leukaemia...	2
full_text	44142	44142	1. A Commentary on Feed...	1
journal	46486	187	cancers	8486
topics	46486	225	[Article]	21516
keywords	46486	40577	[]	3601

Tabella 5.1: Descrizione della struttura e distribuzione del dataset collezionato.

Dall'analisi emerge che, delle 46486 istanze raccolte, 44142 sono uniche. Ulteriori indagini sulle 2344 istanze duplicate hanno rivelato che queste, in realtà, non presentano dati nei campi `doi, pcmid, full_text`. La Tabella 5.1 conferma tale osservazione. Per mantenere la qualità del dataset, si sono quindi eliminate queste istanze con il comando `df.dropna(subset=['full_text'], inplace=True)`.

5.3 Analisi Statistica e Visualizzazione dei Dati

Una volta corrette le anomalie nei dati, l'attenzione si è spostata sull'analisi statistica del dataset, con un occhio di riguardo verso la rilevazione di *outliers*, ovvero quelle istanze che si discostano significativamente dalla norma.

5.3.1 Distribuzione per Rivista Scientifica

Per meglio comprendere la composizione del dataset e facilitarne il caricamento su HuggingFace, suddiviso per rivista, si è stimata la distribuzione delle

```python
journal_counts = articles["journal"].value_counts()

filtered_journal_counts = journal_counts[journal_counts / journal_counts.sum() > 0.01]

mask = articles["journal"].isin(filtered_journal_counts.index)

articles.loc[~mask, "journal"] = "Others"

filtered_article_counts = articles["journal"].value_counts()

plt.figure(figsize=(10, 10))
filtered_article_counts.plot.pie(autopct='%1.1f%%')
plt.axis('equal')
plt.show()
```

Listato 5.5: Codice Python per graficare la distribuzione delle riviste.

istanze per ciascuna pubblicazione. Utilizzando il codice Python riportato nel listato 5.5, sono state filtrate e visualizzate solo le riviste che rappresentano più dell'1% del totale, assegnando le altre alla categoria "Others".

Questo approccio ha prodotto il grafico a torta in Figura 3.1, che illustra la percentuale di istanze per ciascuna rivista. La decisione di escludere le riviste con meno dell'1% delle istanze è stata presa per evitare di avere split del dataset con un numero insufficiente di dati, garantendo così una rappresentazione equilibrata.

5.3.2 Analisi della Frequenza dei Dati

Per valutare la dimensione degli attributi pertinenti agli esperimenti, sono stati realizzati degli istogrammi che evidenziassero le distribuzioni di frequenza di interesse.

Frequenza del numero di caratteri

L'analisi è iniziata esaminando la lunghezza dei testi, misurata dal numero di caratteri per le sezioni `plain_text`, `technical_text` e `full_text`. Si è prodotto un grafico congiunto per i primi due e un grafico separato per il testo completo, a causa delle loro diverse dimensioni.

Il codice utilizzato per visualizzare la distribuzione della lunghezza degli articoli è mostrato nel listato 5.6.

Gli istogrammi risultanti mostrano che il `technical_text` presenta una notevole varianza, con molti valori anomali che vanno da 4000 a 20000 caratteri, mentre il `plain_text` appare più omogeneo. Il `full_text`, invece, evidenzia alcuni valori anomali con oltre 100000 caratteri. Le distribuzioni ottenute sono mostrate in Figure 5.1 per technical e plain, e 5.2 per full.

```python
import matplotlib.pyplot as plt

plain_text_lengths = articles["plain_text"].str.len()
technical_text_lengths = articles["technical_text"].str.len()

plt.figure(figsize=(10, 5))

plt.hist(plain_text_lengths, bins=100, alpha=0.5, label="Plain Text")
plt.hist(technical_text_lengths, bins=100, alpha=0.5, label="Technical Text")

plt.title("Summary and Technical Text Length Distribution")
plt.xlabel("Length")
plt.ylabel("Frequency")
plt.legend()
plt.xlim(0, 20000)
plt.xticks(range(0, 20001, 4000))

plt.show()
```

Listato 5.6: Python script per visualizzare le distribuzioni di frequenza delle lunghezze degli articoli.

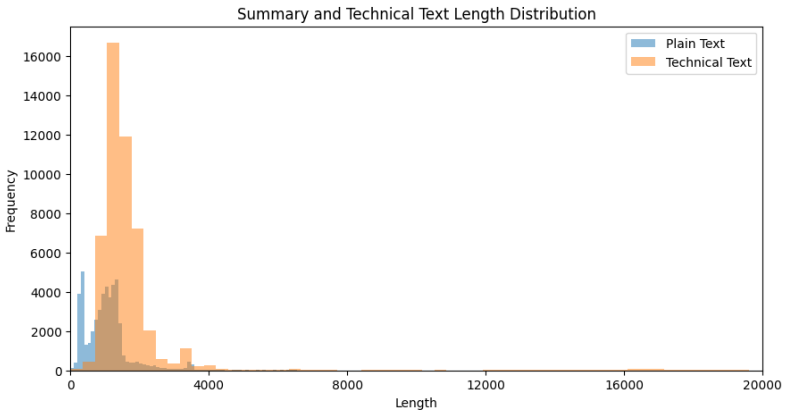


Figura 5.1: Frequenza numero di caratteri per plain e technical text.

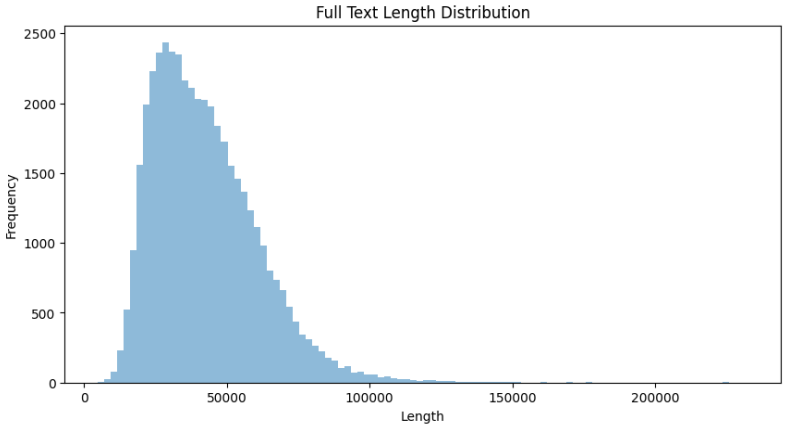


Figura 5.2: Frequenza numero di caratteri per full text.

Sezione	Avg Word Count
Plain Text	144.87
Technical Text	238.61
Full Text	7615.55

Tabella 5.2: Numero medio di parole per ogni sezione di un articolo.

Frequenza del numero di parole

La tokenizzazione, eseguita con `nltk.tokenize`, ha permesso di calcolare il conteggio medio delle parole per sezione, con risultati che corrispondono alle aspettative: il `plain_text` è il più conciso, seguito da `technical_text` e poi da `full_text`, il più esteso. La media del conteggio delle parole è presentata nella Tabella 5.2.

Il codice impiegato è visibile nel listato 5.7.

Per una visualizzazione dettagliata, si sono poi generati istogrammi per il conteggio delle parole utilizzando un codice simile a quello precedentemente descritto (nel listato 5.8). Questi istogrammi hanno confermato i problemi individuati con il `technical_text`, rivelando un numero significativo di outliers con un conteggio parole che oscilla tra 1000 e 3200, a dispetto di una media di circa 238 parole. Analogamente, l'analisi del `full_text` ha svelato alcune istanze sotto le 1000 parole e altre che eccedevano le 20000, sebbene in minor numero.

Come nel caso precedente, si è scelto di presentare un unico script per prevenire ridondanze. Dai grafici risultanti, illustrati nelle Figure 5.3 per il testo semplice e tecnico e nella Figura 5.4 per il testo completo, emergono le distribuzioni di interesse.

```
from nltk.tokenize import word_tokenize
nltk.download('punkt')

articles['plain_word_count'] = articles['plain_text'].apply(lambda text:
↪  len(word_tokenize(text)))
articles['technical_word_count'] = articles['technical_text'].apply(lambda text:
↪  len(word_tokenize(text)))
articles['full_word_count'] = articles['full_text'].apply(lambda text:
↪  len(word_tokenize(text)))

print(articles['plain_word_count'].mean(),
      articles['technical_word_count'].mean(),
      articles['full_word_count'].mean())
```

Listato 5.7: Tokenizzazione delle istanze e calcolo dei valori medi.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))

plt.hist(articles["plain_text_word_count"], bins=100, alpha=0.5, label="Plain Text Words")

plt.hist(articles["technical_text_word_count"], bins=100, alpha=0.5, label="Technical Text
↪  Words")

plt.title("Summary and Technical Text Word Count Distribution")
plt.xlabel("Words Count")
plt.ylabel("Frequency")
plt.legend()

plt.xlim(0, 4000)
plt.xticks(range(0, 4001, 400))

plt.show()
```

Listato 5.8: Script per graficare le distribuzioni di frequenza del numero di parole.

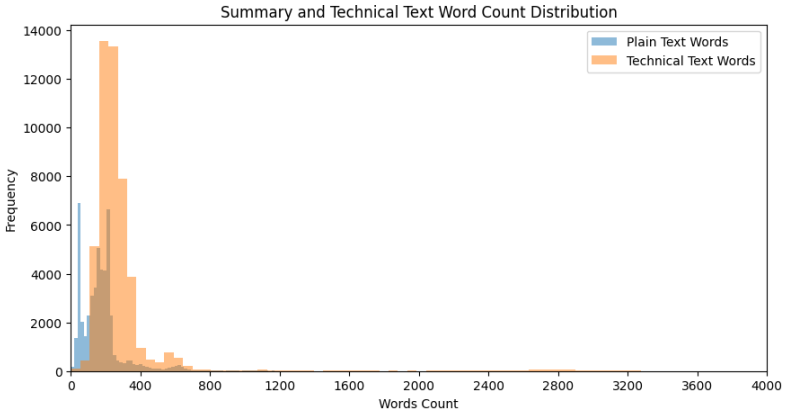


Figura 5.3: Frequenza numero di parole per plain e technical text.

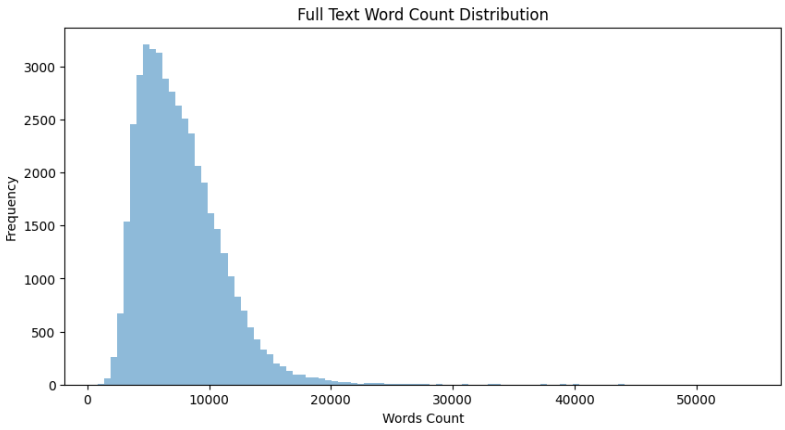


Figura 5.4: Frequenza numero di parole per full text.

Coverage Density e Compression degli articoli

Fino a questo momento ci si è concentrati su caratteristiche inerenti alle varie sezioni prese singolarmente. Un altro fattore molto importante da valutare è anche la relazione che intercorre fra le sezioni di `technical_text`, `plain_text` e `full_text`. In particolare si è deciso di considerare

- **Coverage**. È una misura che riflette la proporzione del testo target (in questo caso `technical_text` o `plain_text`) che è stata ritrovata nel testo source (`full_text`). Essa è concepita per valutare quanto contenuto di un dato testo è coperto o incluso in un altro. Si calcola con

$$\text{Coverage} = \frac{\sum(\text{Match Length})}{\text{Total Length of Target Text}}$$

  dove al numeratore vengono sommate tutte le lunghezze delle sequenze di parole uguali in testo di riferimento e target e al denominatore la lunghezza totale del testo target.

- **Density**. Rappresenta quanto intensamente le corrispondenze tra il testo target e il testo source sono concentrate. Se il testo target ha molte corrispondenze lunghe e ininterrotte nel testo source, la density sarà alta. Formalmente vale

$$\text{Density} = \frac{\sum(\text{Match Length})^2}{\text{Total Length of Target Text}}$$

  Elevare al quadrato la lunghezza di ogni corrispondenza dà più peso a corrispondenze più lunghe e ininterrotte rispetto a molte corrispondenze brevi.

- **Compression**. Rappresenta il rapporto tra la lunghezza del testo source e la lunghezza del testo target. Questa metrica è utile per comprendere quanto il testo target sia più conciso rispetto al testo source. La formula è questa

$$\text{Compression} = \frac{\text{Total Length of Source Text}}{\text{Total Length of Target Text}}$$

  Valori di compressione alti potrebbero indicare che il testo sorgente sia notevolmente più lungo del target, che dunque potrebbe risultare poco rappresentativo. Viceversa, valori inferiori a 1 vorrebbero dire che il target sia inspiegabilmente più lungo del riferimento.

Nella tabella 5.3 è riportato il valore medio per ogni metrica. Si nota che coverage e density risulta di fatto equivalente fra technical e plain, probabilmente

	Plain Text	Technical Text
Coverage	0.9	0.93
Density	3.07	3.67
Compression	78.05	35.04

Tabella 5.3: Coverage Density e Compression degli articoli.

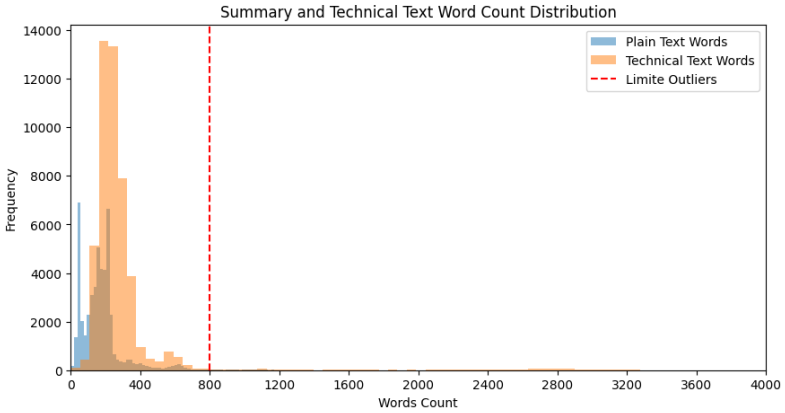


Figura 5.5: Outliers per plain e technical.

dovuto al fatto che in entrambi vi è lo stesso quantitativo di sequenze di parole identiche. Da notare, invece, che la compression è oltre il doppio per plain rispetto a technical, questo fa capire come tipicamente il technical text sia di dimensione molto più elevata rispetto al plain.

Oltre a questo, sono state graficate tramite istogrammi le distribuzioni di ciascuna di queste metriche. Di rilevanza è la compression per cui si sono notate alcune istanze fuori range mostrate in Figura 5.8 e 5.9.

5.4 Rimozione degli outliers

Come anticipato, dai grafici in Figura 5.3, 5.4, emergono degli *outliers* con numero di parole anomale rispetto alla media, sia per `technical_text` che per `full_text`.

Affinché il dataset sia coerente ed omogeneo, si è deciso di scartare questi valori anomali optando per lasciare solo quelli il cui numero di parole non superi le 800 per summary e technical (in quanto le loro frequenze sono quasi equivalenti). Per quanto riguarda full si sono tenute le istanze con parole comprese fra 1000 e 20000.

Le Figure 5.5 e 5.6 mostrano graficamente i valori che sono stati scartati.

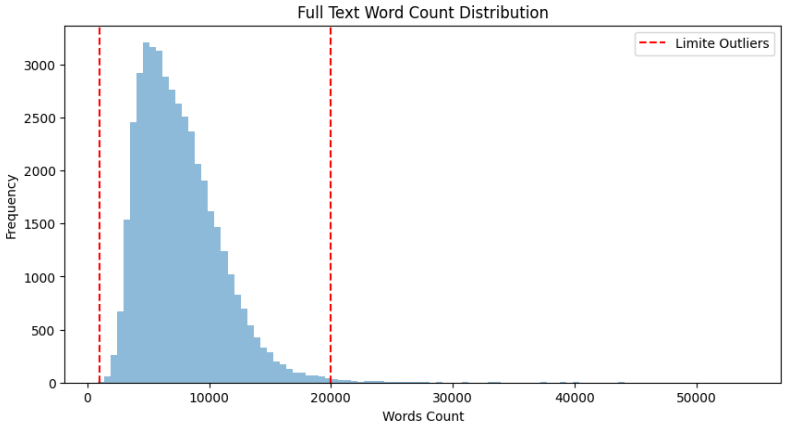


Figura 5.6: Outliers per full text.

```
articles = articles[(articles["technical_text_word_count"] <= 800) &
                    (articles["plain_text_word_count"] <= 800) &
                    (articles["full_text"] >= 1000) &
                    (articles["full_text"] <= 20000)]
```

Listato 5.9: Script per rimozione outliers per numero di parole.

Per farlo si è impiegato il codice mostato nel listato 5.9 che consente, grazie alla potenza di `Pandas`, di scartare tutte le istanze indesiderate in una sola volta.

Questo ha portato a delle distribuzioni più significative che, se plottate, mostrano in maniera più chiara con che frequenza compaino gli articoli aventi un certo numero di parole. Queste sono visibili in Figura 5.7 per plain e tecnhnical. Per `full_text` non è mostrata poiché molto simile a 5.4.

Fatto ciò si sono rimosse le istanze che avessero compression oltre un limite soglia definito su 400 per plain e 135 per technical. Graficamente si vede nelle Figure 5.8, 5.9.

5.5 Caricamento e Organizzazione del dataset su HuggingFace

Il dataset è stato pulito dalle istanze non desiderate e successivamente caricato su HuggingFace. Questo processo ha richiesto la creazione di codice specifico per consentire il caricamento efficace del dataset sul repository remoto. Inoltre, è stato sviluppato del codice aggiuntivo per abilitare gli utenti a

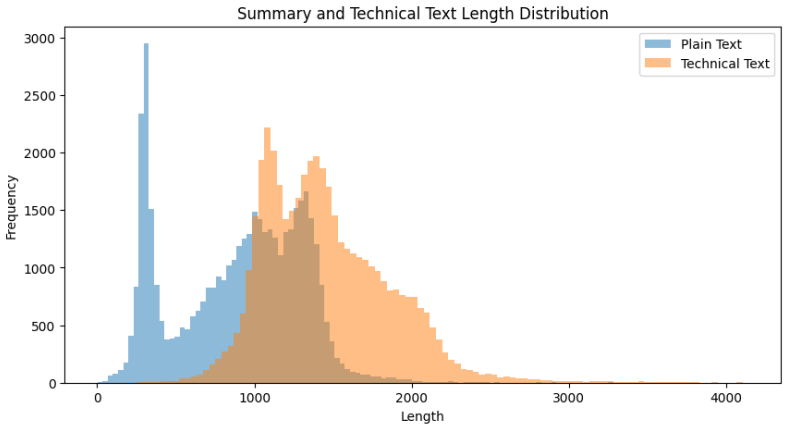


Figura 5.7: Istogramma numero di parole per plain e technical texts post pulizia outliers.

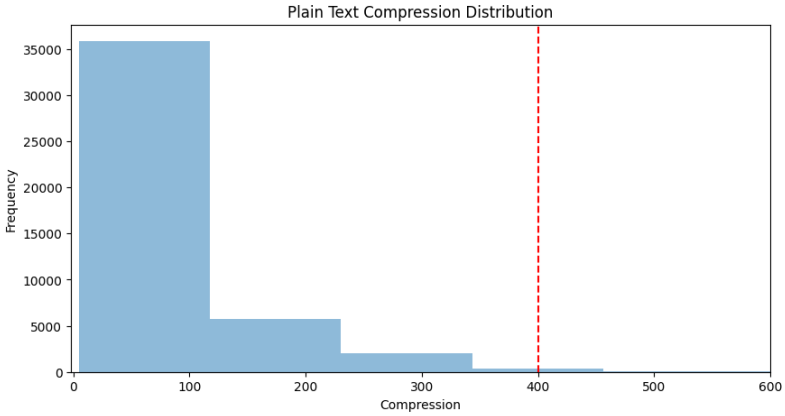


Figura 5.8: Outliers Compression Plain Text.

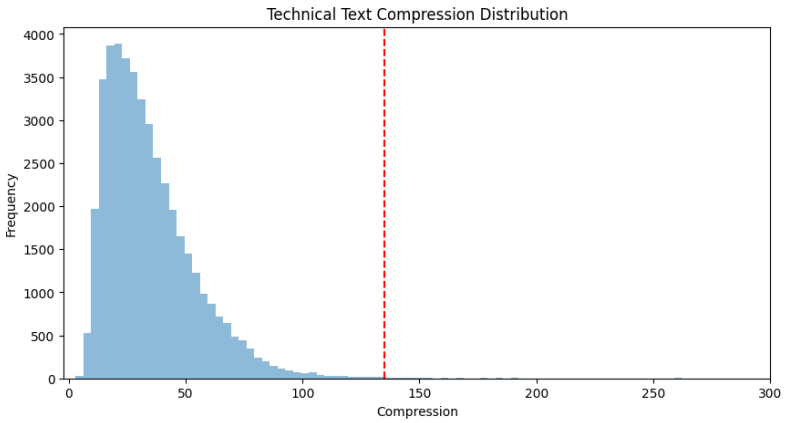


Figura 5.9: Outliers Compression Full Text.

```python
_VERSION = "1.0.0"

_JOURNALS = { ... }

def split_and_save(examples, journal_name):
    train, validate, test = np.split(examples.sample(frac=1, random_state=42),
                                     [int(.8*len(examples)), int(.9*len(examples))])
    for subset, kind in zip([train, validate, test], ['train', 'validation', 'test']):
        subdir = f"sci_lay/data/{_VERSION}/{kind}/{_JOURNALS.get(journal_name, 'OTHER')}"
        save_examples(subset.to_dict(orient="records"), subdir)

def save_examples(examples, output_dir):
    os.makedirs(output_dir, exist_ok=True)
    file_path = os.path.join(output_dir, f"{os.path.basename(output_dir)}.jsonl")
    with open(file_path, "a") as file:
        for example in examples:
            file.write(json.dumps(example) + "\n")

others = []
for journal, examples in dataset.groupby("journal"):
    if journal.lower() in _JOURNALS:
        split_and_save(examples, journal.lower())
    else:
        others.append(examples)
if others:
    other_examples = pd.concat(others, ignore_index=True)
    split_and_save(other_examples, 'OTHER')
```

Listato 5.10: Caricamento del dataset su HuggingFace.

scaricare il dataset tramite l'API di HuggingFace, fornendo loro la flessibilità di selezionare specifici sottoinsiemi di riviste da includere.

5.5.1 Caricamento del Dataset su HuggingFace

Questa fase è stata svolta prendendo come riferimento il dataset BigPatent. Per un caricamento corretto, è necessario organizzare le istanze in file compressi (`.zip`) nominati in base allo split di appartenenza (ad esempio, train, test, validation). All'interno di ciascun file zip, le istanze sono state suddivise in directory specifiche per ogni rivista. Questi file `.zip` devono essere collocati nel percorso `data/version/`. Il codice utilizzato automatizza la divisione dei dati in train, validation e test secondo una proporzione del 80-10-10 e li distribuisce nelle appropriate cartelle. Per categorizzare le riviste sotto l'etichetta "Others", è stato creato un dizionario contenente i nomi delle riviste illustrate nella Figura 3.1, insieme alle relative abbreviazioni. Le istanze appartenenti a riviste non incluse in questo dizionario sono state allocate nella categoria "Others".

Il codice utilizzato è riportato nel Listato 5.10.

```python
from datasets import load_dataset

ds = load_dataset("paniniDot/sci_lay")  # default is 'all' journals
ds = load_dataset("paniniDot/sci_lay", "all")  # the same as above
ds = load_dataset("paniniDot/sci_lay", "NC")  # only 'NC' journal (Nature Communications)
ds = load_dataset("paniniDot/sci_lay", journals=["NC", "A"]) #a subset of the whole
↪  dataset
```

Listato 5.11: Download di una specifica parte del dataset da HuggingFace.

5.5.2 Personalizzazione dell'API Hugging Face per il Recupero di Dataset Specifici per Rivista

Per fare in modo che il dataset possa essere scaricato selezionando una specifica rivista, è necessario personalizzare l'API base che HuggingFace mette a disposizione degli utenti per fare download dei dataset. Nello specifico si voleva ottenere il comportamento mostrato nel Listato 5.11, dove è sufficiente specificare versione e riviste da cui prendere gli articoli, per ottenere la parte richiesta.

Per fare questo, nonostante la documentazione di HuggingFace non fosse molto chiara, si è sviluppata la classe `SciLayConfig`, una configurazione personalizzata per costruire un dataset all'interno del framework `datasets` di HF, appositamente realizzata per *SciLay*. Questa classe estende `datasets.BuilderConfig` (si veda la doc relativa a HuggingFace Builder Classes), che è una classe base usata per definire i parametri per la costruzione di un dataset.

Questa classe è stata progettata per personalizzare il processo di caricamento o costruzione del dataset in modo che gli utenti possano specificare esattamente quali riviste (e quindi quali articoli) vogliono includere nel loro dataset, e in quale versione.

Il passo successivo è stato quello di implementare una classe che si occupasse effettivamente di costruire il dataset. Questa classe, denominata `SciLay` estende `datasets.GeneratorBasedBuilder` pensato proprio per generare dataset a partire da un repository HuggingFace. Per la costruzione del dataset, questa classe fa uso di `SciLayConfig` andando a definire una serie di possibili configurazioni in `BUILDER_CONFIGS`.

La classe presenta tre metodi di utilità ereditati dalla superclasse che li definisce astratti.

1. `_info(self)`. Restituisce un oggetto `DatasetInfo`, il quale incapsula tutte le informazioni del dataset, dunque features e loro tipi, descrizione del dataset, licenza, ecc.

```python
import datasets

_JOURNALS = { ... }

_VERSION = "1.0.0"

class SciLayConfig(datasets.BuilderConfig):
    """BuilderConfig for SciLay."""

    def __init__(self, journals="all", version=_VERSION, **kwargs):
        """BuilderConfig for SciLay.
        Args:
            journals (str or list, default 'all'): List of journal names. Either 'all' or
            ↪  a combination of {'NC',..., 'OTHER'}.
            **kwargs: keyword arguments forwarded to super.
        """
        if isinstance(journals, str):
            journals = [journals]
        name = "+".join(journals)
        if name == "all":
            journals = list(_JOURNALS)
        if version != _VERSION:
            name = f"{name}-{version}"
        super().__init__(name=name, version=version, **kwargs)
        self.journals = journals
```

Listato 5.12: Classe SciLayConfig per ottenere i dataset divisi per rivista.

2. `_split_generator(self, dl_manager)`. Ritorna un generatore per ciascuno split del dataset (ad esempio, train, validation, test). Utilizza il gestore di download per scaricare e estrarre i dati dagli URL specificati, che sono formattati con la versione del dataset e il nome dello split. Restituisce una lista di oggetti `SplitGenerator` che contengono le informazioni per generare ciascuno split.

3. `_generate_examples(self, paths=None)`. Genera esempi del dataset. Itera attraverso i percorsi forniti e apre ogni file, legge ogni riga (che si presume sia un oggetto JSON) e produce un esempio del dataset. Ogni esempio è un dizionario che mappa i nomi delle caratteristiche ai loro valori.

Il codice della classe è mostrato nel listato 5.13.

5.6 Esperimenti su SciLay con Baseline

Terminata questa fase il progetto è entrato nel vivo, cominciando a sperimentare varie baseline astrattive di modelli progettati per fare summarization.

```python
import json
import os

_SPLIT_NAMES = {datasets.Split.TRAIN: "train", datasets.Split.VALIDATION: "validation", datasets.Split.TEST: "test"}
_URL = "data/{version}/{split_name}.zip"

class SciLay(datasets.GeneratorBasedBuilder):
    """SciLay datasets."""

    BUILDER_CONFIG_CLASS = SciLayConfig
    BUILDER_CONFIGS = [
        SciLayConfig(journals="all", description="Articles from all journals."),
    ] + [
        SciLayConfig(journals=k, description=f"Articles from journals {k}: {v}")
        for k, v in sorted(_JOURNALS.items())
    ]
    DEFAULT_CONFIG_NAME = "all"
    VERSION = _VERSION

    def _info(self):
        return datasets.DatasetInfo(
            description=_DESCRIPTION,
            features=datasets.Features({
                _DOI: datasets.Value("string"),
                ...
                _KEYWORDS: datasets.Sequence(datasets.Value("string"))
            }),
            supervised_keys=(_FULL_TEXT, _SUMMARY),
            homepage=_HOMEPAGE,
            license=_LICENSE,
            citation=_CITATION,
        )

    def _split_generators(self, dl_manager):
        """Returns SplitGenerators."""
        urls = {
            split: _URL.format(version=self.config.version, split_name=split_name)
            for split, split_name in _SPLIT_NAMES.items()
        }
        dl_paths = dl_manager.download_and_extract(urls)
        paths = {
            split: [
                dl_manager.iter_files(os.path.join(dl_paths[split], split_name, code)) for code in self.config.journals
            ]
            for split, split_name in _SPLIT_NAMES.items()
        }
        return [
            datasets.SplitGenerator(name=split, gen_kwargs={"paths": paths[split]})
            for split in _SPLIT_NAMES
        ]

    def _generate_examples(self, paths=None):
        """Yields examples."""
        for paths_per_journal in paths:
            for path in paths_per_journal:
                with open(path, "rb") as fin:
                    for row in fin:
                        json_obj = json.loads(row)
                        yield json_obj[_DOI], {
                            _DOI: json_obj[_DOI],
                            ...
                            _KEYWORDS: json_obj[_KEYWORDS]
                        }
```

Listato 5.13: Classe SciLay per scaricare i dataset.

5.6.1 Tipologia di Esperimenti Condotti

Sono stati condotti numerosi esperimenti di generazione di riassunti sul dataset proposto. Fra i principali vengono riportati i seguenti:

- *Addestramento Completo su Seq2Seq Models*: In questa fase, si è proceduto all'addestramento, alla validazione e al test di diversi modelli Seq2Seq sull'intero dataset. L'obiettivo era valutare l'efficacia di questi modelli nel generare riassunti, confrontando i risultati ottenuti con le prestazioni teoriche descritte dai loro sviluppatori. In particolare, l'attenzione si è concentrata sulla capacità dei modelli di produrre sia lay summary (riassunti per un pubblico non specialistico) che abstract accademici partendo dagli articoli completi.

- *Transfer Learning Few-Shot con il Modello Migliore*: Dopo aver identificato il modello Seq2Seq più performante nella fase precedente, questo è stato ulteriormente addestrato in un contesto di few-shot learning. In pratica, il modello è stato addestrato su un subset limitato di dati (tipicamente una sola rivista), per poi essere testato sulla sua capacità di generalizzare su altri subset del dataset. Questo esperimento mirava a valutare quanto efficacemente il modello potesse adattarsi e mantenere alte prestazioni anche con una quantità limitata di dati di addestramento.

- *Valutazione Zero-Shot con Large Language Models (LLM)*: In questa categoria di esperimenti, alcuni Large Language Models sono stati valutati in un contesto zero-shot. Ciò significa che al modello è stato fornito un articolo e delle linee guida specifiche per generare un riassunto, senza un addestramento preventivo sul tipo specifico di riassunto richiesto. L'obiettivo era valutare la qualità dei riassunti generati dal modello in assenza di addestramento specifico, per comprendere la sua capacità di adattamento e generazione di testi in condizioni non ottimali.

Questi esperimenti hanno fornito una panoramica sull'efficacia di varie tecniche di NLP nel contesto della generazione automatica di riassunti, offrendo spunti importanti per future ricerche e applicazioni pratiche nel campo.

5.6.2 Configurazione Ambiente di Sviluppo

Come anticipato nel capitolo 4, l'uso del server ha richiesto l'utilizzo di containers Docker per fare in modo di mantenere tutte le dipendenze congelate all'interno di un ambiente riproducibile sui vari calcolatori a disposizione nel cluster. Poiché i vari esperimenti hanno richiesto librerie differenti, si è reso necessario creare differenti container docker, ciascuno con le specifiche dipendenze.

Dockerfile per Modelli Seq2Seq

Nel listato 5.14 è mostrato il Dockerfile utilizzato per fare esperimenti su modelli Seq2Seq come T5, Bart Large, LSG Bart Large, ecc. Per esso, come anticipato, si parte da un'immagine messa a disposizione da HuggingFace per avere già a disposizione molti dei pacchetti che sono stati impiegati. Fra i requirements più importanti spiccano

- *bitsandbytes==0.41.1*: Essenziale per il caricamento di modelli in precisione ridotta, questa caratteristica è cruciale per l'utilizzo di modelli di grandi dimensioni che, senza di essa, non potrebbero essere addestrati su GPU con 24GB di VRAM.
- *codecarbon==1.2.0*. Utilizzato per monitorare l'impatto ambientale in termini di CO2 prodotta degli esperimenti condotti.
- *ConfigParser==6.0.0*. Utilizzato per gestire le configurazioni e le impostazioni. Consente di avere degli oggetti che contengano tutte le configurazioni passate al codice python tramite file sh in un formato organizzato.
- *datasets==2.10.0*. Libreria fornita da HuggingFace per scaricare i dataset dal loro repository.
- *transformers==4.31.0*. Fornisce un'interfaccia semplice e flessibile per utilizzare modelli ad architettura transformers per una varietà di applicazioni NLP, come classificazione del testo, risposta a domande, traduzione automatica, ecc. Fra i modelli pre addestrati offerti spiccano Bart, T5, vari modelli LSG, ecc.
- *wandb==0.15.10*. Fornisce strumenti per registrare e visualizzare metriche, output e modelli durante l'addestramento. È molto utilizzato per il tracciamento degli esperimenti, la gestione dei modelli e l'analisi dei risultati.
- *torch==2.0.1*. Framework di calcolo scientifico, ampiamente usata in applicazioni di machine learning, che offre ampio supporto per algoritmi di ottimizzazione e reti neurali. È nota per la sua efficienza e facilità d'uso.

Dockerfile per LLM

Per gli esperimenti zero-shot effettuati su Large Language Models (LLM), è stato cruciale l'impiego di un ambiente ottimizzato per i driver CUDA, al fine

```
FROM huggingface/transformers-pytorch-latest-gpu

# Set the working directory inside the container
WORKDIR /workspace/

# Copy the requirements file into the container at /workspace
COPY requirements.txt /workspace/

# Install the required packages
RUN pip install --no-cache-dir -r requirements.txt
```

Listato 5.14: Dockerfile per Baseline Seq2Seq.

```
FROM nvcr.io/nvidia/pytorch:22.12-py3

# Set the working directory inside the container
WORKDIR /workspace/

# Update linux packages and python packet manager
RUN apt-get update -y && apt-get autoremove -y
RUN pip install --upgrade pip

# Copy the requirements file into the container at /workspace
COPY requirements_llm.txt /workspace/

# Install the required packages
RUN pip install --no-cache-dir -r requirements_llm.txt

RUN pip install torch --index-url https://download.pytorch.org/whl/cu118

RUN pip install pyg_lib torch_scatter torch_sparse torch_cluster torch_spline_conv -f
↪  https://data.pyg.org/whl/torch-2.1.0+cu118.html

RUN pip uninstall transformer_engine -y
```

Listato 5.15: Dockerfile per LLM.

di sfruttare appieno le GPU Nvidia disponibili. A questo scopo, il Dockerfile mostrato nel listato 5.15 è stato creato specificamente per soddisfare queste esigenze. La struttura è molto simile a quella impiegata per i Seq2Seq, tuttavia `PyTorch` è stato installato da un repository specifico che lo supporta per CUDA 18 (ultima versione retrocompatibile con GPU Nvidia GeForce RTX 3090), assicurando così la compatibilità e l'efficienza nell'uso delle risorse di calcolo.

Poiché il pacchetto `Transformer Engine` non funziona in retrocompatibilità, viene disinstallato per non generare errori in fase di esecuzione.

5.6.3 Esperimenti con Modelli Seq2Seq

Dopo aver configurato adeguatamente l'ambiente di lavoro, è stato realizzato un codice versatile per addestrare, validare e testare diversi modelli sul dataset creato. Invece di modificare il codice per ogni nuovo modello, si è optato per una soluzione più flessibile e generica: le impostazioni specifiche di addestramento vengono passate direttamente come argomenti a linea di comando. Questo metodo semplifica il processo sperimentale e assicura una codebase uniforme per ogni esperimento.

Tale strategia facilita notevolmente l'automatizzazione, permettendo l'addestramento di più modelli con un singolo script. Il listato 5.16 illustra l'uso di un ciclo for che consente di passare attraverso diversi modelli, subset del dataset SciLay o altre configurazioni di addestramento. Sebbene il listato non mostri tutti i parametri utilizzati per ragioni di spazio, gli esempi forniti evidenziano la capacità dello script di adattarsi a svariate impostazioni. Questo include la scelta del modello, la lunghezza massima dei testi, il numero di epoche, il tasso di apprendimento e altri parametri cruciali.

In conclusione, questo approccio elimina la necessità di scrivere codici differenti per ogni modello o dataset, risparmiando tempo e riducendo sia la complessità che il rischio di errori.

Addestramento e Valutazione sull'Intero Dataset

Come anticipato, in questa prima fase si sono usati diversi modelli Seq2Seq sull'intero dataset per verificarne l'aderenza con le prestazioni teoriche descritte dai creatori. Nel caso in cui si eguagliassero o addirittura superassero si dimostrerebbe la qualità del dataset prodotto. Alcuni dei modelli impiegati non sono stati addestrati per mancanza di risorse hardware. Malgrado gli sforzi impiegati per ridurre la dimensione dei modelli, abbassando la precisione e il numero di token in input, la VRAM a disposizione non si è rivelata sufficiente. Il codice impiegato è suddiviso in

- Definizione degli Argomenti per effettuare l'addestramento. Questa fase è fondamentale per stabilire i parametri di configurazione del modello e del processo di addestramento.

- Controllo di eventuali checkpoint di modelli precedentemente addestrati. Questa fase garantisce l'efficienza dell'addestramento, permettendo di riprendere da un punto salvato in precedenza. Ciò evita la necessità di ricominciare l'addestramento da capo, risparmiando tempo e risorse computazionali.

```bash
#!/bin/bash

N=4
JOURNALS=("A" "B" "PLGEN" "MBIO")

for ((i=0; i<N; i++)); do
    CUDA_VISIBLE_DEVICES=0 python3 run_generation.py \
        --logging online \
        --do_train \
        --do_eval \
        --do_predict \
        --dataset_name paniniDot/sci_lay \
        --subset_name ${JOURNALS[$i]} \
        --task_name full_to_technical_summarization \
        --model_name_or_path ccdv/lsg-bart-large-4096 \
        --log_level error \
        --gradient_accumulation_steps 1 \
        --max_source_length 4096 \
        --max_target_length 512 \
        --max_eval_samples 10 \
        --generation_max_length 512 \
        --num_train_epochs 4 \
        --learning_rate 5e-5 \
        --save_strategy epoch \
        --evaluation_strategy epoch \
        --gradient_checkpointing \
        --load_best_model_at_end \
        --predict_with_generate \
        --metric_for_best_model eval_rouge1 \
        --save_total_limit 1 \
        --num_beams 5 \
        --generation_num_beams 2 \
        --group_by_length \
        --sortish_sampler \
        --weight_decay 0.01 \
        --label_smoothing_factor 0.1 \
        --include_inputs_for_metrics \
        --remove_unused_columns \
        --per_device_train_batch_size 1 \
        --per_device_eval_batch_size 1 \
        --fp16 \
        ...
    wait
done
```

Listato 5.16: Script sh per addestramento di un modello su molteplici subset di SciLay.

- Preprocessing del Dataset. Una fase cruciale dove i dati vengono preparati per l'addestramento. Questo include la selezione delle colonne appropriate dal dataset e la loro tokenizzazione, un passo essenziale per trasformare i dati grezzi in un formato che il modello può comprendere ed elaborare.

- Configurazione dell'Ottimizzatore, Scheduler e Trainer ed Effettuazione dell'Addestramento. Questo è il cuore del processo di addestramento, dove vengono impostati l'ottimizzatore e lo scheduler per controllare il modo in cui il modello apprende durante l'addestramento. Il trainer coordina tutto il processo, gestendo il ciclo di addestramento e di validazione.

- Salvataggio dei risultati. Dopo l'addestramento, è importante salvare i risultati, compresi i pesi del modello e le metriche di performance. Questa fase prevede semplicemente la scrittura su file delle metriche calcolate in fase di inferenza.

Definizione degli Argomenti La sezione iniziale del codice si occupa della definizione delle classi `DataTrainingArguments` e `ModelArguments`, che specificano rispettivamente le opzioni per i dati in input e per il modello da addestrare, come illustrato nel listato 5.17.

Queste classi sono implementate sfruttando la libreria `dataclasses` che consente, tramite l'uso di *decoratori* Python, di arricchire le classi create dagli utenti con metodi speciali per una gestione efficace degli attributi. Ad esempio con l'ausilio della funzione `field()` che permette di dettagliare ulteriormente gli attributi, fornendo controlli automatici durante l'esecuzione del programma. Ad esempio si possono specificare *default*, ovvero un valore di default per un dato attributo. Inoltre, si fa uso della classe `Optional` per indicare che determinati attributi possono essere omessi, semplificando la gestione di scenari in cui si potrebbero altrimenti utilizzare molteplici condizioni if o valori `None`.

In aggiunta, il listato 5.17 dimostra l'uso pratico di queste classi. Vengono create istanze per `DataTrainingArguments` e `ModelArguments`, nonché per `Seq2SeqTrainingArguments`, una classe fornita dal modulo `transformers`, contenente tutti i parametri necessari per addestrare un modello Seq2Seq. Queste istanze sono poi passate come argomenti a `HfArgumentParser`, una classe progettata per semplificare il parsing degli argomenti dalla riga di comando in script focalizzati sull'uso di modelli di machine learning.

Caricamento di eventuali checkpoint Come mostra il listato 5.18, la funzione per il controllo del checkpoint fa uso della funzione `get_last_checkpoint` di `transformers`. Questa funzione è appositamente realizzata per caricare i pesi determinati in un precedente addestramento di uno stesso modello. La

```python
from dataclasses import dataclass, field
from typing import Optional
from transformers import HfArgumentParser, Seq2SeqTrainingArguments

@dataclass
class DataTrainingArguments:
    """
    Arguments pertaining to what data we are going to input our model for training and
    ↪ eval.
    """
    dataset_name: Optional[str] = field(
        default=None, metadata={"help": "The name of the dataset to use (via the datasets
        ↪ library)."}
    )
    ...

@dataclass
class ModelArguments:
    """
    Arguments pertaining to which model/config/tokenizer we are going to fine-tune from.
    """
    model_name_or_path: str = field(
        metadata={"help": "Path to pretrained model or model identifier from
        ↪ huggingface.co/models"}
    )
    ...

def main():
    parser = HfArgumentParser((ModelArguments, DataTrainingArguments,
    ↪ Seq2SeqTrainingArguments))
    model_args, data_args, training_args = parser.parse_args_into_dataclasses()
    ...
```

Listato 5.17: Definizione delle classi per gli argomenti.

```python
from transformers.trainer_utils import get_last_checkpoint

def check_for_last_checkpoint(training_args, logger):
    if os.path.isdir(training_args.output_dir):
        if training_args.do_train and not training_args.overwrite_output_dir:
            last_checkpoint = get_last_checkpoint(training_args.output_dir)
            if last_checkpoint is None and len(os.listdir(training_args.output_dir)) > 0:
                raise ValueError(...)
            elif last_checkpoint is not None and training_args.resume_from_checkpoint is
            ↪ None:
                logger.info("checkpoint detected")
            return last_checkpoint
```

Listato 5.18: Controllo di eventuali checkpoint salvati.

funzione `check_for_last_checkpoint` può tornare un modello valido o `None` qualora non si sia trovato.

Preprocessing del Dataset La fase di preprocessing del dataset è fondamentale per rendere i dati compatibili e comprensibili per il modello di machine learning. Questo processo si articola in diversi passaggi critici (di cui elencati solo i principali per ragioni di sintesi):

- *Caricamento del Dataset*: Inizia con il caricamento dei dati attraverso la funzione `load_dataset()` del modulo `datasets`.
- *Estrazione delle Colonne Rilevanti*: Utilizzando le funzioni `get_dataset_columns` e `get_text_and_summary_columns`, vengono estratte le colonne di input (`text_column`) e di riferimento (`summary_column`) dal dataset. Questo permette di identificare quali dati saranno usati per l'addestramento e per la valutazione del modello.
- *Tokenizzazione e Pulizia dei Dati*: Il passo finale è la tokenizzazione delle colonne di input e riferimento, un processo eseguito dalla funzione `preprocess_dataset`. Questa fase trasforma il testo in un formato che il modello può processare efficacemente, rimuovendo contemporaneamente le colonne non necessarie per semplificare il dataset.

Per ragioni di chiarezza, nel listato 5.19 sono illustrate solo alcune delle funzioni principali utilizzate in questa fase, evidenziando come vengono invocate nel main. È importante notare come il codice, nelle righe 45 a 56, sia strutturato in modo simmetrico per le fasi di valutazione (eval) e test, garantendo un approccio coerente e sistematico nella preparazione dei diversi set di dati.

Configurazione di Ottimizzatore, Scheduler e Trainer ed Effettuazione dell'Addestramento Questa fase rappresenta il nucleo dell'intero esperimento, comprendendo l'addestramento, la validazione e il test del modello selezionato. L'approccio adottato prevede un ciclo di addestramento durante il quale, al termine di ogni epoca, si effettua una sessione di validazione per valutare le prestazioni del modello. Una volta completato l'addestramento, si esegue un'ulteriore sessione di validazione seguita dal ciclo di test. Come per la fase di preprocessing del dataset, anche questa procedura si articola in vari passaggi:

- *Configurazione dell'ottimizzatore.* L'ottimizzatore è un componente critico nell'addestramento dei modelli di machine learning, il cui compito è minimizzare l'errore del modello aggiornando i suoi parametri (equivalente della discesa del gradiente descritta nel capitolo 2 di questa tesi). Tra i

```python
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
import logging
import os
import sys

def preprocess_function(examples, text_column, summary_column, tokenizer, max_source_length, max_target_length):
    inputs, targets = zip(*((i, t) for i, t in zip(examples[text_column], examples[summary_column]) if i and t))

    model_inputs = tokenizer(list(inputs), max_length=max_source_length, truncation=True)
    labels = tokenizer(list(targets), max_length=max_target_length, truncation=True)

    model_inputs["labels"] = labels["input_ids"]
    return model_inputs

def preprocess_dataset(dataset, preprocess_fn, column_names, overwrite_cache, desc, training_args, text_column, summary_column, tokenizer, max_source_length, max_target_length):
    with training_args.main_process_first(desc=f"{desc} map pre-processing"):
        return dataset.map(
            lambda examples: preprocess_fn(
                examples,
                text_column=text_column,
                summary_column=summary_column,
                tokenizer=tokenizer,
                max_source_length=max_source_length,
                max_target_length=max_target_length,
            ),
            batched=True,
            remove_columns=column_names,
            load_from_cache_file=not overwrite_cache,
            desc=f"Running tokenizer on {desc} dataset",
        )

def main():
    ...
    raw_datasets = load_dataset(
        data_args.dataset_name,
        data_args.subset_name,
        cache_dir=model_args.cache_dir,
        download_mode="force_redownload",
        use_auth_token=True if model_args.use_auth_token else None,
    )

    tokenizer = AutoTokenizer.from_pretrained(model_args.model_name_or_path)
    text_column, summary_column = get_text_and_summary_columns(data_args, get_dataset_columns(raw_datasets, training_args), task_name_mapping)

    if training_args.do_train:
        train_dataset = preprocess_dataset(
            raw_datasets["train"],
            preprocess_function,
            column_names,
            data_args.overwrite_cache,
            "train",
            training_args,
            text_column,
            summary_column,
            tokenizer,
            data_args.max_source_length,
            data_args.max_target_length
        )
```

Listato 5.19: Preprocessing del dataset.

diversi algoritmi di ottimizzazione disponibili, per questo esperimento è stato scelto AdamW [25].

- *Configurazione dello Scheduler*: Lo scheduler è un elemento chiave che modula il tasso di apprendimento (learning rate) durante l'addestramento. In questa fase, si configura lo scheduler per specificare come il learning rate dovrebbe variare attraverso le diverse epoche, aiutando così il modello a convergere più efficacemente verso una soluzione ottimale.

- *Definizione del Trainer e Addestramento*: Il trainer è lo strumento che gestisce l'addestramento del modello. Viene utilizzata la classe `Seq2SeqTrainer` per impostare un ambiente di addestramento che incorpora il modello, l'ottimizzatore e lo scheduler. Questa classe permette una gestione efficiente e personalizzabile del processo di addestramento, integrando aspetti critici come il calcolo della loss, la valutazione delle metriche e la gestione dei dati.

Il listato 5.20 riassume i passaggi più importanti di questa fase.

Risultati Nelle tabelle 5.4 e 5.5 sono riportati rispettivamente i risultati di ogni addestramento per il task di summarization da full a lay e da full a technical. Le tabelle sono divise per tipologia di modello (lineare o quadratico) e presentano tutte le metriche descritte nel capitolo 5.

I modelli sono stati tutti quanti addestrati 4 epoche (in quanto attraverso test preliminari si è visto statisticamente che il picco di loss è raggiunto in quest'epoca, consentendoci di risparmiare tempo) e con i seguenti iperparametri (specificati solo i più rilevanti)

- Numero di token di input: 1024 per BART Large e Base, 4096 per tutti gli altri

- Numero di token generati in output: 512

- Learning Rate di partenza $5 \cdot 10^{-5}$

- Batch Size di Training 1

- Batch Size di Validation 1

- Precisione mista floating point 16 bit.

Come si evince dalla tabella 5.4, per quanto riguarda il task di generazione di riassunto lay

```python
import torch
from transformers import Seq2SeqTrainer, get_scheduler

def setup_optimizer(model, weight_decay, learning_rate):
    no_decay = ["bias", "LayerNorm.weight", "layer_norm.weight"]
    optimizer_grouped_parameters = [
        {
            "params": [p for n, p in model.named_parameters() if not any(nd in n for nd in no_decay)],
            "weight_decay": weight_decay,
        },
        {
            "params": [p for n, p in model.named_parameters() if any(nd in n for nd in no_decay)],
            "weight_decay": 0.0,
        },
    ]
    optimizer = torch.optim.AdamW(optimizer_grouped_parameters, lr=learning_rate)
    return optimizer

def setup_scheduler(optimizer, num_update_steps_per_epoch, num_train_epochs):
    max_train_steps = num_train_epochs * num_update_steps_per_epoch
    return get_scheduler(
        name="linear",
        optimizer=optimizer,
        num_warmup_steps=0,
        num_training_steps=max_train_steps
    )

def main():
    ...
    optimizer = setup_optimizer(model, training_args.weight_decay, training_args.learning_rate)
    lr_scheduler = setup_scheduler(optimizer, len(train_dataloader), training_args.num_train_epochs)
    optimizers = (optimizer, lr_scheduler)

    trainer = Seq2SeqTrainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset if training_args.do_train else None,
        eval_dataset=eval_dataset if training_args.do_eval else None,
        tokenizer=tokenizer,
        data_collator=data_collator,
        compute_metrics=lambda eval_preds: compute_metrics(eval_preds, tokenizer=tokenizer) if training_args.predict_with_generate else None,
        optimizers=optimizers,
    )

    #checkpoint loaded with check_for_last_checkpoint
    train_result = trainer.train(resume_from_checkpoint=checkpoint)
```

Listato 5.20: Configurazione optimizer, scheduler e trainer e addestramento.

- BART Large mostra le migliori performance in Rouge-1, Rouge-2 e Rouge-L, suggerendo una buona capacità di catturare sia i dettagli chiave (Rouge-2) che la globalità del testo originale (Rouge-L). Oltre a questo presenta il miglior $\mathcal{R}$, suggerendo una maggiore varietà nel riassunto, perdendosi tuttavia nelle altre metriche.
- Per quanto riguarda LSG BART Large, invece, risulta leggermente inferiore a BART Large nelle metriche Rouge, pur superandolo leggermente nelle metriche BERT Score, BLEU-1, BLEU-2, BLEU-3 e BLEU-4, suggerendo una qualità generale leggermente superiore.
- Passando poi a LSG Pegasus Large si nota che ottiene i migliori BARTScore F,P pur essendo estremamente bassi per tutti i modelli.
- Terminando la valutazione con LSG BART Base si nota che presenta prestazioni complessivamente inferiori rispetto agli altri modelli, il che potrebbe essere attribuito al suo minor numero di parametri.

Per quanto riguarda il task di generazione di abstract, la tabella 5.5 mostra come

- BART Large mostri buone performance pur essendo superato da altri modelli in quasi tutte le metriche, indicando una possibile inefficacia per compiti di generazione di testi tecnici.
- LSG BART Large risulti essere migliore in Rouge-1, Rouge-2 e Rouge-L e $\mathcal{R}$, suggerendo eccellenza nel catturare dettagli chiave e significato globale.
- LSG Pegasus Large: mostri miglioramenti significativi in diverse metriche rispetto al task lay, pur rimanendo inferiore a LSG BART Large in molte di esse.
- Infine LSG BART Base mostri miglioramenti in BERT Score e BLEU rispetto al task lay, suggerendo un'adattabilità maggiore ai contenuti tecnici.

Confrontando i due task emerge come i modelli tendano a mostrare prestazioni leggermente migliori sul task tecnico rispetto a quello lay in termini di qualità generale (BertScore, BLEU). Questo potrebbe suggerire che i modelli sono più adatti a gestire contenuti tecnici, forse a causa di una maggiore coerenza e specificità nel linguaggio tecnico, nonché adesione al testo originale.

In conclusione, LSG BART Large e LSG BART Base sembrano essere i modelli più consistenti su entrambi i task, con LSG BART Large leggermente migliore nel complesso. Per questo motivo è il modello impiegato nel task successivo di transfer learning.

Model	R-1	R-2	R-L	$\mathcal{R}$	bertscore	BS-F	BS-P	BS-R	B-1	B-2	B-3	B-4	Υ
Quadratic													
BART LARGE	**40.24**	**12.61**	**29.99**	**27.26**	42.22	-3.152	-3.109	-3.195	35.74	18.70	11.11	7.26	**55.29**
Linear													
LSG BART LARGE	39.54	12.28	29.45	26.75	**42.3**	-3.216	-3.292	**-3.14**	**35.76**	**18.82**	**11.39**	**7.61**	53.79
LSG PEGASUS LARGE	36.44	10.72	27.85	24.72	37.60	**-3.041**	**-2.79**	-3.292	27.89	14.24	8.51	5.65	54.61
LSG BART BASE	39.0	11.91	29.19	26.37	41.56	-3.249	-3.329	-3.169	35.42	18.51	11.16	7.44	54.57

Tabella 5.4: Risultati dei Modelli per il Task di Summarization da Full a Lay.

Model	R-1	R-2	R-L	$\mathcal{R}$	bertscore	BS-F	BS-P	BS-R	B-1	B-2	B3	B-4	Υ
Quadratic													
BART LARGE	41.49	13.41	29.64	27.81	44.44	**-3.147**	**-3.127**	-3.168	**37.7**	20.18	12.1	7.94	**54.94**
Linear													
LSG BART LARGE	**42.81**	**14.49**	**30.68**	**28.94**	**45.72**	**-3.147**	-3.188	**-3.106**	37	**20.53**	**12.75**	**8.6**	54.86
LSG PEGASUS LARGE	40.33	13.25	29.48	27.35	42.87	-2.921	-2.699	-3.143	31.61	17.48	11.07	7.66	53.49
LSG BART BASE	41.65	13.58	29.84	27.99	45.05	-3.169	-3.224	-3.114	36.81	20.25	12.53	8.45	55.52

Tabella 5.5: Risultati dei Modelli per il Task di Summarization da Full a Technical.

Osservazioni Pratiche Come precedentemente menzionato, la varietà di modelli considerati per questi task è notevolmente più ampia di quella mostrata nei risultati. Tuttavia, a causa delle limitazioni imposte dalle GPU a disposizione e malgrado i numerosi tentativi di ottimizzazione, vari modelli non si sono rivelati abbastanza efficienti in termini di risorse per essere utilizzati negli esperimenti. Tra questi, i modelli che hanno presentato maggiori difficoltà sono stati T5 (nelle versioni Large e X-Large), PEGASUS, PEGASUS-X e LED (versione Large), PRIMERA. I problemi incontrati sono stati principalmente legati alla capacità della VRAM, pari a 24GB. Per cercare di aggirare queste limitazioni, abbiamo adottato diverse strategie, tra cui:

- La riduzione del numero di token in input (fino a 1024).
- La diminuzione del batch size per training e validation (ridotto a 1, non è possibile spingersi oltre).
- L'impiego di un Optimizer a 8 bit, prelevato dalla libreria `bitsandbytes`, in sostituzione del classico AdamW a 32 bit.

Queste misure, pur essendo tentativi validi di ottimizzazione, non sono state sufficienti a rendere questi modelli praticabili con le risorse a disposizione.

Transfer Learning

Come precedentemente menzionato, il modello Seq2Seq più efficace, identificato come LSG Bart Large, è stato sottoposto a un ulteriore processo di addestramento nel contesto del few-shot learning. Questo approccio comporta

la selezione di un numero molto ristretto di documenti appartenenti ad una specifica rivista, costituendo così un dataset limitato. Successivamente, il modello è stato addestrato e validato utilizzando questo insieme ristretto di dati. L'obiettivo principale era di testare il modello su articoli provenienti da altre riviste per valutare la sua capacità di adattarsi a diversi tipi di articoli, stili e linguaggi, nonostante la limitata quantità di dati di addestramento a disposizione. In questo studio di tesi, gli esperimenti si sono focalizzati esclusivamente sulla produzione di riassunti laici.

Per quanto riguarda l'implementazione, il codice utilizzato è molto simile a quello impiegato negli esperimenti precedenti. Tuttavia, vi sono state alcune modifiche significative nel modo in cui il dataset è stato caricato e come sono state gestite le fasi di addestramento, validazione e test. Queste variazioni sono state necessarie per adattare il processo al contesto specifico del few-shot learning.

Caricamento del dataset La prima differenza fondamentale rispetto al codice usato negli esperimenti con l'intero dataset riguarda il metodo di caricamento del dataset dalla libreria HuggingFace. Nel caso precedente, il caricamento era gestito direttamente nel main, precisamente alla riga 35 del listato 5.19, attraverso l'uso della funzione standard `load_dataset` di `datasets`. Tuttavia, per gli esperimenti correnti, è necessario adottare un approccio differente per caricare e gestire separatamente i diversi dataset, al fine di eseguire in modo efficace le fasi di training, validazione e test su specifiche porzioni di essi. L'utilizzo diretto della funzione `load_dataset('paniniDot/sci_lay',['A', 'B', ...])` non è adeguato in questo contesto, poiché produce un dataset unificato attraverso il merge dei journal specificati, il che non è compatibile con gli obiettivi di questo esperimento. Per risolvere questa problematica, come illustrato nel listato 5.21, è stata implementata la funzione `get_datasets`. Questa funzione genera tre dizionari, ognuno contenente coppie chiave-valore. La chiave corrisponde al nome di un specifico subset (ad esempio, una rivista), mentre il valore è un oggetto `Dataset` formato nel seguente modo:

In questi dizionari sono racchiuse le istanze per il corrispondente split di ogni subset selezionato. Questa soluzione consente di gestire separatamente i vari subset per le fasi di addestramento, validazione e test all'interno dell'esperimento.

Preparazione del dataset Nel contesto attuale, il processo di addestramento e validazione si focalizza esclusivamente sul subset specificato come argomento, mentre il test viene eseguito su tutti i subset. Ciò determina un approccio di preprocessing dei dataset leggermente diverso rispetto a quello adottato negli esperimenti precedenti. La funzione descritta nel listato 5.21 genera

```python
import datasets

def get_datasets(dataset_name, cache_dir, use_auth_token, train_subset, subsets):
    train, validation, test = {}, {}, {}

    for subset in subsets:
        ds = load_dataset(dataset_name, subset, cache_dir=cache_dir,
        ↪  use_auth_token=use_auth_token)
        if subset == train_subset:
            train[subset] = ds['train']
            validation[subset] = ds['validation']
        test[subset] = ds['test']

    return train, validation, test
```

Listato 5.21: Caricamento dei Dataset per Transfert Learning Few-Shot.

```python
def main():
    subsets = ["A", "B", "C", ...]
    train_dataset_dict, eval_dataset_dict, test_dataset_dict =
    ↪  get_datasets(data_args.dataset_name, model_args.cache_dir,
    ↪  model_args.use_auth_token, data_args.subset_name, subsets)
    ...
    train_dataset = preprocess_dataset(
        train_dataset_dict[data_args.subset_name],
        ...
    )
    for subset_name, test_dataset in test_dataset_dict.items():
        column_names = test_dataset.column_names
        text_column, summary_column = get_text_and_summary_columns(data_args, column_names,
        ↪  task_name_mapping)
        test_dataset_dict[subset_name] = preprocess_dataset(
            test_dataset,
            ...
        )
```

Listato 5.22: Preparazione dei Dataset per Transfert Learning Few-Shot.

dizionari per le fasi di training e validazione, contenenti un singolo elemento corrispondente al subset passato come argomento dallo script sh. Per quanto riguarda il test, invece, avrà un'istanza per ogni subset del dataset. Questo comporta che la preparazione dei dataset andrà fatta per ciascuna istanza di ogni dataset. Questo avviene mediante un ciclo che esegue la preparazione una volta per ogni subset di test. Quanto detto è mostrato nel listato 5.22. Il codice per eval è identico a quello di train, pertanto viene omesso per risultare più conciso.

Addestramento Validazione e Test Cosi come per il preprocessing, anche le fasi di addestramento, validazione e test del modello sono gestite con un

```
from transformers import Seq2SeqTrainer,AutoTokenizer
import json
import os

def main():
    trainer = Seq2SeqTrainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=eval_dataset,
        tokenizer=tokenizer,
        data_collator=data_collator,
        compute_metrics=lambda eval_preds: compute_metrics(eval_preds,
        ↪  tokenizer=tokenizer) if training_args.predict_with_generate else None,
        optimizers=optimizers,
    )

    #train + eval
    predict_and_save_results(trainer, last_checkpoint, train_dataset, len(train_dataset),
    ↪  training_args, tokenizer, "train")

    #test
    for subset_name, test_dataset in test_dataset_dict.items():
        predict_and_save_results(trainer, last_checkpoint, test_dataset, len(test_dataset),
        ↪  training_args, tokenizer, "test", subset_name)
```

Listato 5.23: Esecuzione di addestramento, validazione e test.

approccio ciclico che consente di processare ogni subset definito. Questo processo è illustrato nel listato 5.23. Per ottimizzare e rendere più efficiente il codice, le operazioni comuni a tutte e tre le fasi sono state incapsulate nella funzione `predict_and_save_results`. Questa funzione, dopo aver determinato la fase corrente (train, eval, o test), esegue l'addestramento o la predizione a seconda del contesto. Inoltre, calcola vari score e li archivia in un file, il cui percorso è specificato come argomento della funzione. In particolare, durante la fase di test, la funzione si assicura che i risultati siano salvati in una sotto cartella dedicata al subset specifico su cui il test è stato eseguito.Questa funzione, dopo aver determinato la fase corrente (train, eval, o test), esegue l'addestramento o la predizione a seconda del contesto. Inoltre, calcola vari score e li archivia in un file, il cui percorso è specificato come argomento della funzione. In particolare, durante la fase di test, la funzione si assicura che i risultati siano salvati in una sotto cartella dedicata al subset specifico su cui il test è stato eseguito.

Risultati Per ogni rivista utilizzata nell'addestramento del dataset, il modello viene testato su tutte le altre riviste, generando le metriche di valutazione citate nel capitolo 4, per ciascun test. Di seguito sono presentati i punteggi medi ottenuti addestrando il modello su una specifica rivista e testandolo su tutte le altre.

Dall'analisi della tabella 5.6 emergono alcuni spunti di riflessione interessanti. In particolare i risultati complessivi sono discretamente buoni, ma tendono a essere inferiori rispetto all'addestramento sul dataset completo di tutte le riviste. Questo è prevedibile, poiché l'addestramento su un singolo journal limita la varietà e la quantità di esempi di addestramento, peggiorandone la qualità.

Analizzando più nel dettaglio i risultati si evince che la rivista *PLoS Pathogens* (PLPAT) sembra eccellere, ottenendo i punteggi più alti in Rouge-1, Bleu-1, Bleu-2 e Bleu-3. Questo suggerisce che i modelli addestrati su articoli di questa rivista potrebbero avere una capacità superiore di catturare sia i dettagli chiave che la globalità del testo originale. Questa rivista ottiene punteggi sorprendentemente più alti di *Cancers* (C) che si colloca in seconda posizione eccellendo in Rouge-2, Rouge-L e $\mathcal{R}$, pur avendo, come mostra la tabella 3.1, un numero di istanze di addestramento notevolmente superiore a PLPAT. Questo fatto potrebbe essere contestualizzato valutando la qualità dei documenti di addestramento di C, probabilmente inferiore rispetto a quella degli articoli di PLPAT. Per quanto riguarda la miglior comprensione semantica degli articoli *PLoS Genetics* emerge su tutti gli altri con un bertscore di 36.19.

Al contrario, le riviste *Scientific Data* (SD) e *Biology* (B) mostrano i risultati più deboli, probabilmente a causa della loro limitata quantità di dati. Questo evidenzia l'importanza della dimensione del dataset di training nella generalizzazione del modello su vari dataset.

I risultati suggeriscono che un numero maggiore di istanze di training tende generalmente a migliorare la capacità di generalizzazione del modello, ma non vi è una correlazione diretta e proporzionale. Riviste con un numero di istanze inferiore a C, come PLPAT e PLGEN, hanno superato C in diverse metriche cruciali. Di conseguenza, l'addestramento su PLPAT, PLGEN o C sembra essere particolarmente efficace. In conclusione, la scelta della rivista per l'addestramento influenza notevolmente i risultati sulle altre riviste e oltre al numero di istanze, è essenziale considerare anche la varietà, la specificità e la qualità dei dati. Riviste con più istanze possono mostrare prestazioni migliori, ma ciò non esclude che dataset ben curati, anche se più piccoli, possano produrre risultati notevoli. Ciò sottolinea l'importanza di selezionare dati di addestramento adeguati per ottimizzare l'efficacia del modello su dataset diversificati.

5.6.4 Esperimenti su LLM

Terminati gli esperimenti su modelli sequence to sequence, l'attenzione si è spostata su modelli LLM per confrontarne i risultati con quelli ottenuti da modelli con meno parametri. A causa dei vincoli di tempo, ci si è limitati a

Training su	R-1	R-2	R-L	$\mathcal{R}$	bertscore	BS-F	BS-P	BS-R	B-1	B-2	B-3	B-4	Υ
NC	30.12	8.50	24.92	20.99	33.67	-3.27	-2.91	-3.63	7.19	3.74	2.22	1.39	2.79
A	35.08	8.87	25.42	22.83	34.99	-3.45	-3.37	-3.53	26.18	12.74	6.94	4.17	5.77
PLGEN	34.98	8.79	25.21	22.69	**36.19**	-3.40	-3.35	**-3.45**	27.41	13.41	7.37	**4.48**	9.21
PLPAT	**35.35**	9.04	25.54	23.01	35.94	-3.34	-3.18	-3.50	**28.36**	**13.83**	**7.49**	4.44	4.55
PLCB	34.49	8.43	24.90	22.32	35.31	-3.44	-3.40	-3.48	27.19	13.04	7.07	4.24	**9.91**
PLNTD	34.21	8.40	24.85	22.20	34.39	-3.50	-3.50	-3.50	26.78	12.88	6.87	4.05	6.87
B	24.07	3.30	21.10	16.00	12.20	-3.35	**-2.48**	-4.21	12.00	4.58	1.67	0.68	6.46
I	34.70	8.70	25.44	22.65	34.29	-3.34	-3.11	-3.58	23.88	11.63	6.23	3.64	6.20
PLB	35.08	8.50	25.26	22.64	34.97	**-3.25**	-2.91	-3.60	26.64	12.90	6.70	3.75	3.34
CB	30.32	7.33	23.29	20.11	31.43	-3.35	-3.02	-3.69	9.53	4.59	2.36	1.30	6.63
SD	9.85	2.61	12.11	8.17	11.00	-4.77	-5.69	-3.85	2.48	1.44	0.97	0.72	9.73
MBIO	34.79	8.31	24.93	22.38	34.66	-3.33	-3.10	-3.58	26.83	12.83	6.65	3.71	4.85
C	35.34	**9.41**	**25.67**	**23.18**	35.52	-3.44	-3.43	**-3.45**	24.31	12.01	6.87	4.38	5.12
OTHER	33.89	8.70	25.19	22.33	33.84	-3.40	-3.23	-3.56	22.35	10.97	6.07	3.69	1.73

Tabella 5.6: Risultati medi esperimento di transfer learning.

compiere esperimenti zero shot, questo significa che gli LLM non sono stati addestrati, bensì, basandosi sulla loro versione non fine-tuned, si è fornito un prompt della forma

Summarize the following document according to these guidelines: {guidelines}. Document: {document}

aspettandosi che il modello completi tale input con un riassunto valido. Le direttive utilizzate sono quelle fornite dagli autori del sito Plain Language Summaries, disponibili esclusivamente su richiesta scritta.

Inferenza Zero Shot su Rivista mBio

Cosi come fatto per i modelli Seq2Seq, anche in questo caso sono stati condotti degli esperimenti preliminari su un dataset di dimensioni limitate. Questo consente, fra le altre cose, di determinare un lower bound per i risultati raggiungibili e le tempistiche necessarie a compiere un addestramento più consistente. In particolare, in questo caso, la scelta del dataset è ricaduta sul subset di SciLay della rivista rivista mBio, avendo in totale 759 istanze diviste in train, validation e test.

Il codice impiegato risulta essere molto semplice poiché privo della fase di addestramento. Come nel contesto dei modelli Seq2Seq, anche in questo caso si è adottato un approccio parametrizzabile via script .sh, facilitando l'esecuzione di molteplici esperimenti senza dover modificare il codice python.

Il sorgente python si suddivide in

- Caricamento di dataset, modello e tokenizer a partire dalle classi di utilità di `datasets` e `transformers`.
- Inferenza sul dataset fornendo ogni istanza e le linee guida al modello.

Caricamento di Dataset Modello e Tokenizer Il codice, per questa parte, risulta molto simile a quello impiegato nei modelli Seq2Seq. L'unica cosa in cui differiscono è la classe scelta per caricare il modello. In particolare, come mostra il listato 5.24, nel caso di large language models, `transformers` offre la classe `AutoModelForCasualLM` piuttosto che `AutoModelForSeq2SeqLM`. La differenza chiave tra questi modelli è che `AutoModelForSeq2SeqLM` è orientato a compiti come traduzione e riassunto, usando un approccio encoder-decoder, mentre `AutoModelForCasualLM` è utilizzato per compiti di generazione di testo e previsione di parole successive, generalmente basandosi su un'architettura avente unicamente un encoder.

```
import datasets
from transformers import AutoModelForCausalLM, AutoTokenizer

def main():
    raw_datasets = load_dataset(args.dataset_name, args.dataset_subset)
    train_dataset = raw_datasets[args.split]
    references = train_dataset["plain_text"]

    tokenizer = AutoTokenizer.from_pretrained(args.model)
    ll_model = AutoModelForCausalLM.from_pretrained(args.model, cache_dir="../llms",
    ↪  load_in_4bit=True, trust_remote_code=True, device_map="auto")
```

Listato 5.24: Caricamento di dataset, tokenizer e modello LLM.

Inferenza La fase d'inferenza si basa sulla funzione `generate_summary`. Questa inizia ricevendo il modello, il tokenizer, un articolo e la lunghezza massima dell'input per l'inferenza. L'input viene tokenizzato e fornito al modello per generare un riassunto. Trattandosi di un modello generativo che fornisce in output il più probabile completamento del prompt fornito in input, si utilizza un token speciale [END] per distinguere tra l'input e l'output generato. Dopo aver decodificato l'output, la funzione separa e restituisce solo quest'ultima parte, escludendo l'input iniziale. Il token [END] viene riconosciuto dal modello dopo essere stato aggiunto richiamando sul tokenizer il metodo `add_special_tokens` e ridimensionando il modello con `resize_token_embeddings`. La funzione per generare i riassunti si ripete per ogni documento del dataset passato, calcolando per ciascuno un punteggio che mira a valutare la qualità dell'output generato. Quanto detto viene parzialmente mostrato nel listato 5.25.

Risultati Come anticipato sono stati utilizzati due modelli LLM, Llama2 7B Chat e Mistral 7B Instruct, per condurre gli esperimenti. Entrambi sono varianti ottimizzate per compiti conversazionali delle loro versioni base general-purpose, specializzate per rispondere a prompt testuali in un contesto di chat. Per valutarne le prestazioni, sono stati condotti test utilizzando input di 4096

```python
import torch

def generate_summary(document, guidelines, tokenizer, ll_model, max_source_length, device):
    instruction = f"Summarize the following document according to these guidelines: {guidelines}. Document: {document}"
    inputs = tokenizer(instruction, return_tensors="pt", max_length=max_source_length, truncation=True).to(device)

    new_token = torch.tensor([[tokenizer.additional_special_tokens_ids[0]]]).to(device)

    input_ids_tensor = torch.cat((inputs['input_ids'], new_token), dim=1)

    summary_ids = ll_model.generate(input_ids=input_ids_tensor, max_length=512)
    dec_out = tokenizer.batch_decode(summary_ids, skip_special_tokens=False)
    return dec_out[0].split('[END]')[1]
...
def main():
    ...
    tokenizer.add_special_tokens({'additional_special_tokens': [" [END]"]})
    ll_model.resize_token_embeddings(len(tokenizer))

    predictions = []
    with torch.no_grad():
        for document in tqdm(train_dataset["full_text"]):
            summary = generate_summary(document, args.guidelines, tokenizer, ll_model, args.max_source_length, device)
            predictions.append(summary)

    results = compute_metrics(predictions, references, device)

```

Listato 5.25: Processo di inferenza su LLM.

token e di 200 token. Dai risultati, sorprendentemente, è emerso che i prompt limitati a 200 token hanno registrato punteggi mediamente superiori, come evidenziato dalla tabella 5.7. A causa di risultati preliminari insoddisfacenti, gli esperimenti si sono concentrati esclusivamente sulla generazione di riassunti laici, escludendo il task di produzione di abstract.

Modello	R-1	R-2	R-L	$\mathcal{R}$	BS	BS-F	BS-P	BS-R	B1	B2	B3	B4	Υ
Llama2 Chat 7B													
200 Tokens	**28.15**	**4.50**	**21.62**	**17.91**	**-14.86**	**-3.402**	**-2.997**	**-3.806**	**22.44**	**8.32**	**3.29**	**1.37**	**47.37**
4096 Tokens	0.33	0.0	0.7	0.34	-95.19	-6.154	-7.772	-4.535	0.0	0.0	0.0	0.0	8.47
Mistral Instruct 7B													
200 Tokens	19.15	2.99	16.3	12.75	-24.1	-4.334	-4.666	-4.003	8.28	3.11	1.33	0.56	40.88
4096 Tokens	0.19	0.0	0.3	0.12	-105.11	-10.34	-11.21	-7.55	0.0	0.0	0.0	0.0	7.62

Tabella 5.7: Risultati dei Modelli di Chat Llama2 7B e Mistral Instrument 7B per Diverse Lunghezze di Token.

La tabella evidenzia la superiorità di Llama2 Chat in entrambi gli esperimenti rispetto a Mistral Instruct, suggerendo una minore efficacia di quest'ultimo

in task di summarization di testi lunghi. Un aspetto notevole è la netta differenza di prestazioni tra i riassunti da 200 token e quelli da 4096 token. In entrambi i modelli, i punteggi per i riassunti più lunghi sono drasticamente inferiori, indicando una possibile perdita di efficacia in presenza di testi di grandi dimensioni.

Conclusioni e sviluppi futuri

Questo lavoro ha rappresentato per me l'inizio di uno studio approfondito sul mondo del Natural Language Processing e più in generale sulle tecniche più all'avanguardia del deep learning.

La creazione di SciLay rappresenta un contributo importante per la comunità scientifica, in quanto fornisce una risorsa preziosa per lo sviluppo e la valutazione di modelli di intelligenza artificiale in grado di sintetizzare contenuti biomedici complessi in formati più comprensibili.

Gli esperimenti condotti con modelli Seq2Seq e Large Language Models (LLM) hanno rivelato risultati relativamente modesti, indicando che la sfida di generare riassunti laici (ma anche tecnici) da articoli scientifici biomedici rimane un problema aperto, confermato anche dal fatto che in letteratura non esistono molte ricerche in merito. Questo aspetto, tuttavia, non deve essere visto negativamente; al contrario, evidenzia un'opportunità stimolante per ulteriori ricerche nel settore del Natural Language Processing (NLP). Infatti, il lavoro svolto sta già contribuendo alla stesura di un articolo destinato alla prestigiosa rivista ACL, dimostrando l'importanza e la rilevanza del progetto nel campo dell'NLP.

Da un punto di vista comparativo, i modelli Seq2Seq hanno mostrato prestazioni superiori rispetto agli LLM testati, suggerendo una minore idoneità di questi ultimi, in assenza di uno specifico addestramento, per l'obiettivo specifico di generare riassunti laici. Questa osservazione apre la strada a indagini future su possibili miglioramenti e adattamenti dei modelli Seq2Seq e LLM analizzati, nonché a un'esplorazione più approfondita di altri modelli esistenti.

Per quanto riguarda gli obiettivi futuri, sono previsti, relativamente all'articolo menzionato in precedenza, diversi percorsi di ricerca. In primo luogo, c'è l'intenzione di sviluppare nuove architetture Seq2Seq che possano migliorare ulteriormente i risultati attuali. In secondo luogo, si prevede di testare altri modelli Seq2Seq già esistenti, per valutare la loro efficacia nel contesto specifico del dataset SciLay. Infine, per gli LLM, è prevista la sperimentazione con nuovi modelli, come Zephyr 7B, e modelli con un numero maggiore di parametri, come Llama2 70B e Falcon 180B. Un'area particolarmente promettente riguarda non

solo l'utilizzo di questi modelli per l'inferenza, ma anche la loro applicazione in scenari di addestramento, per esplorare se e come possono essere adattati e ottimizzati per questa specifica sfida di summarization.

In conclusione, questo lavoro di tesi non solo fornisce un contributo significativo alla comunità scientifica con la creazione del dataset SciLay, ma apre anche numerose strade per future indagini e sviluppi nel campo dell'NLP, particolarmente in relazione alla comprensione e alla sintesi di testi scientifici biomedici per un pubblico non specialistico.

Ringraziamenti

Desidero esprimere il mio profondo apprezzamento e gratitudine nei confronti del mio relatore, Prof. Gianluca Moro, e correlatori, il Dott. Paolo Italiani e il Dott. Luca Ragazzi. Il loro supporto, guida competente e consigli preziosi nei mesi di lavoro che hanno preceduto la realizzazione di questa tesi sono stati fondamentali per il suo successo. La loro disponibilità e il clima amichevole e professionale che hanno saputo creare hanno reso questo percorso non solo produttivo, ma anche piacevole e stimolante.

Un particolare ringraziamento va anche al Dott. Giacomo Frisoni, per la sua disponibilità e il sostegno tecnico fornito a qualsiasi ora del giorno e della notte. La sua abilità nel risolvere i problemi più disparati relativi al server è stata fondamentale per il successo di questo lavoro.

Il mio debito verso la mia famiglia è più grande che verso chiunque altro. Essi sono stati le colonne su cui ho potuto poggiare in ogni momento. Mi hanno guidato, ispirato, incoraggiato e sostenuto. Soprattutto hanno sempre creduto in me. Questo lavoro è dedicato a loro.

Vorrei rivolgere un ringraziamento speciale al mio team di gaming "P3tY eSports", compagni straordinari sia nella vita reale (IRL) che sul nostro canale Discord "pussy abusers". Le innumerevoli ore trascorse insieme giocando a Fortnite e Minecraft hanno reso le mie giornate decisamente più leggere durante il periodo intenso di lavoro sulla mia tesi. Questi momenti di chilling e le risate condivise sia online che offline hanno avuto un impatto fondamentale sul mio benessere, permettendomi di affrontare lo stress e la pressione con una maggiore serenità. La vostra amicizia e il sostegno, in ogni forma, hanno giocato un ruolo cruciale nel raggiungimento di questo importante traguardo accademico. Grazie di cuore a tutti voi per essere stati parte essenziale di questo viaggio.

Un ringraziamento altrettanto sentito va agli amici del gruppo "Predapio" dell'università. La vostra presenza ha reso i lunghi periodi di studio più leggeri e piacevoli, e per questo vi sarò sempre grato. La nostra amicizia e le esperienze

condivise rimarranno con me come uno dei ricordi più preziosi di questo periodo.

Desidero esprimere un ringraziamento particolare a Eleonora, per il suo inestimabile supporto in questo periodo delicato. La tua vicinanza e comprensione hanno reso il percorso meno arduo, illuminando i giorni più difficili. La tua presenza costante e il tuo incoraggiamento sono stati fondamentali nel superare le sfide e nel celebrare i successi con rinnovato entusiasmo.

Desidero esprimere il mio più sincero apprezzamento e ringraziamento all'Azienda Agricola 2 Compagnie di Fano per l'ospitalità e le esperienze memorabili offerte durante la mia visita. La qualità eccellente delle salsicce, l'ottimo vino e l'opportunità unica di guidare un trattore hanno reso la mia esperienza veramente eccezionale. Questi momenti trascorsi nella vostra azienda non solo hanno arricchito il mio palato, ma hanno anche contribuito a creare ricordi indimenticabili. La vostra accoglienza e generosità sono state notevoli e hanno lasciato un'impressione duratura. Grazie per avermi accolto con tanta cordialità e per aver condiviso con me la passione e l'impegno che caratterizzano il vostro lavoro.

In ultimo, ma non per importanza, desidero rivolgere un ringraziamento speciale a Riccardo. La tua presenza come spalla affidabile durante l'intero percorso della triennale è stata di inestimabile valore. Studiare insieme, affrontare e superare ogni ostacolo ha giocato un ruolo cruciale nel mio sviluppo accademico e personale. La tua costante disponibilità, il tuo incrollabile sostegno e la tua sincera amicizia sono stati elementi chiave nel raggiungimento di questo importante traguardo.

Bibliografia

[1] Gerd Gigerenzer. *Perché l'intelligenza umana batte ancora gli algoritmi*. Scienza e Idee. Raffaello Cortina Editore, 2023.

[2] N.N. Taleb. *Il cigno nero. Come l'improbabile governa la nostra vita*. Saggi. Tascabili. Il Saggiatore, 2009.

[3] Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.

[4] Angwin Julia, Larson Jeff, Mattu Surya, and Kirchner Lauren. Machine bias. 2016.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[6] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[7] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461, 2019.

[8] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019.

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen

Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[10] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[11] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015.

[12] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[14] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[15] Mahima Pushkarna, Andrew Zaldivar, and Oddur Kjartansson. Data cards: Purposeful and transparent dataset documentation for responsible ai, 2022.

[16] Charles Condevaux and Sébastien Harispe. Lsg attention: Extrapolation of pretrained transformers to long sequences, 2022.

[17] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. PEGASUS: pre-training with extracted gap-sentences for abstractive summarization. *CoRR*, abs/1912.08777, 2019.

[18] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor

Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[19] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

[20] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[21] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In Pierre Isabelle, Eugene Charniak, and Dekang Lin, editors, *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[22] Gianluca Moro, Luca Ragazzi, and Lorenzo Valgimigli. Carburacy: Summarization models tuning and comparison in eco-sustainable regimes with a novel carbon-aware accuracy. In Brian Williams, Yiling Chen, and Jennifer Neville, editors, *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 14417–14425. AAAI Press, 2023.

[23] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert, 2020.

[24] Weizhe Yuan, Graham Neubig, and Pengfei Liu. Bartscore: Evaluating generated text as text generation, 2021.

[25] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.