ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria
Corso di Laurea in Ingegneria e Scienze Informatiche

ABSTRACTIVE LONG DOCUMENT SUMMARIZATION: STUDIO E SPERIMENTAZIONE DI MODELLI GENERATIVI RETRIEVAL-AUGMENTED

Elaborato in
Programmazione Di Applicazioni Data Intensive

Relatore
Prof. Gianluca Moro

Co-relatore
Dott. Ing. Luca Ragazzi

Presentata da
Veronika Folin

Seconda Sessione di Laurea
Anno Accademico 2020 – 2021

PAROLE CHIAVE

Natural Language Processing
Abstractive Summarization
Legal Analytics
Language Models
Deep Learning

A mia madre, che mi è stata vicina trasmettendomi
tanta forza e determinazione, e a mio padre,
che desidero rendere orgoglioso dei traguardi raggiunti.

Abstract

In questa tesi si trattano lo studio e la sperimentazione di un modello generativo retrieval-augmented, basato su Transformers, per il task di Abstractive Summarization su lunghe sentenze legali. La sintesi automatica del testo (Automatic Text Summarization) è diventata un task di Natural Language Processing (NLP) molto importante oggigiorno, visto il grandissimo numero di dati provenienti dal web e banche dati. Inoltre, essa permette di automatizzare un processo molto oneroso per gli esperti, specialmente nel settore legale, in cui i documenti sono lunghi e complicati, per cui difficili e dispendiosi da riassumere. I modelli allo stato dell'arte dell'Automatic Text Summarization sono basati su soluzioni di Deep Learning, in particolare sui Transformers, che rappresentano l'architettura più consolidata per task di NLP. Il modello proposto in questa tesi rappresenta una soluzione per la Long Document Summarization, ossia per generare riassunti di lunghe sequenze testuali. In particolare, l'architettura si basa sul modello RAG (Retrieval-Augmented Generation), recentemente introdotto dal team di ricerca Facebook AI per il task di Question Answering. L'obiettivo consiste nel modificare l'architettura RAG al fine di renderla adatta al task di Abstractive Long Document Summarization. In dettaglio, si vuole sfruttare e testare la memoria non parametrica del modello, con lo scopo di arricchire la rappresentazione del testo di input da riassumere. A tal fine, sono state sperimentate diverse configurazioni del modello su diversi esperimenti e sono stati valutati i riassunti generati con diverse metriche automatiche.

Indice

1 Tecniche per il Machine Learning **1**
1.1 Creazione di un modello 1
1.2 Rete Neurale 5
1.2.1 Storia delle reti neurali 6
1.2.2 Reti neurali odierne 7
1.3 Deep Learning 8
1.4 Transfer Learning 9

2 Natural Language Processing **11**
2.1 Cos'è il NLP? 11
2.2 Tecniche di pre-processing 12
2.3 Rappresentazione del testo 13
2.3.1 One-hot Encoding 13
2.3.2 Modello n-gram 13
2.3.3 Bag of words 13
2.3.4 Word embedding 14
2.4 Applicazioni 15
2.5 Tecnologie di Deep Learning 15
2.5.1 Recurrent Neural Network 16
2.5.2 Convolutional Neural Network 18
2.5.3 Encoder-Decoder 19
2.5.4 Attention Mechanism 20
2.5.5 Transformer 22

3 Automatic Text Summarization **25**
3.1 Extractive Summarization 26
3.2 Abstractive Summarization 27
3.3 Metriche di valutazione 31
3.4 Multi-document Summarization 33
3.4.1 Classificazione delle tecniche 34
3.4.2 Sviluppi futuri 39

4 Progetto **41**
4.1 Introduzione e obiettivi . . . 41
4.1.1 Differenza fra modelli parametrici e non-parametrici . . . 42
4.1.2 Formulazione della soluzione . . . 42
4.2 Panoramica delle architetture . . . 42
4.2.1 RAG . . . 42
4.2.2 BERT . . . 44
4.2.3 BART . . . 46
4.3 Analisi del dataset . . . 46
4.4 Esperimenti . . . 49
4.4.1 Discussione dei risultati . . . 51
4.5 Codice prodotto . . . 53
4.5.1 Preparazione del dataset . . . 53
4.5.2 Modifica del RagRetriever . . . 54
4.5.3 Pre-processamento dei dati . . . 57
4.5.4 Training . . . 58
4.5.5 Evalutation . . . 60
4.6 Tecnologie utilizzate . . . 62

Conclusioni e sviluppi futuri **65**

Ringraziamenti **67**

Bibliografia **69**

Elenco delle figure

1.1 Rappresentazione del fitting di un modello. 2
1.2 Sviluppo delle performance in base al numero di dati. 3
1.3 Punto di arresto anticipato per evitare overfitting. 3
1.4 Struttura di un neurone biologico. 5
1.5 Modello iniziale di neurone artificiale. 6
1.6 Struttura di una rete neurale semplice. 7
1.7 Struttura di un neurone artificiale. 7
1.8 Gerarchia delle discipline di Intelligenza Artificiale. 8

2.1 Esempio di estrazione delle relazioni tra diversi termini. 14
2.2 Neurone di una rete RNN attraverso il tempo. 16
2.3 Struttura di una cella LSTM. 17
2.4 Struttura di una cella GRU. 17
2.5 Convolutional Neural Network. 18
2.6 Architettura WaveNet. 19
2.7 Modello encoder-decoder applicato alla traduzione. 19
2.8 Modello encoder-decoder con Concatenative Attention. 20
2.9 Creazione delle matrici *queries* (Q), *keys* (K) e *values* (V) nel meccanismo di attenzione. 21
2.10 Calcolo degli output nel meccanismo di attenzione. 22
2.11 Architettura del modello Transformer. 23
2.12 Architettura del Multi-Head Attention layer. 24
2.13 Architettura del Scaled Dot-Product Attention layer. 24

3.1 Classificazione delle tecniche di abstractive summarization. . . . 28
3.2 La lunghezza del documento influenza il fattore di ridondanza. . 33
3.3 Tassonomia delle tecniche di multi-document summarization. . . 34
3.4 Architettura dei sistemi ibridi per la summarization. 35
3.5 Strategie di design per sistemi MDS. 36
3.6 Schema del funzionamento di un Variational Auto-Encoder. . . . 37

4.1 Architettura del modello RAG. 43
4.2 Input e output dell'architettura BERT. 45

4.3 Funzionamento dell'architettura BART. 46
4.4 Strategie di mascheramento dell'input. 46
4.5 Distribuzione delle lunghezze dei testi nel dataset *BillSum* rispetto al numero di caratteri, parole e frasi. 47
4.6 Esempio di istanza del dataset *BillSum*. 48
4.7 Interfaccia della piattaforma Google Colaboratory. 63

Capitolo 1

Tecniche per il Machine Learning

Il Machine Learning (ML), o apprendimento automatico, è un insieme di tecniche che permettono di estrarre conoscenza in modo automatico da un insieme di dati, restituendo un modello ottimale che generalizza il comportamento e le caratteristiche del dominio di studio. L'apprendimento automatico risulta particolarmente utile quando la programmazione di un algoritmo, appositamente sviluppato per la risoluzione di un problema, diventa insostenibile in termini di tempo e di costi. Per esempio, si applica nei seguenti casi:

- Classificazione delle immagini.
- Sistema di raccomandazione.
- Filtro anti-spam.
- Motore di ricerca.
- Sistema domotico.

1.1 Creazione di un modello

L'obiettivo del Machine Learning è progettare algoritmi in grado di generare previsioni accurate partendo da task complessi. A tal fine, vi è la fase di addestramento (*training*), che serve ad estrarre un modello di conoscenza partendo da un insieme di dati di esempio (*training examples*). Da questo insieme di dati di addestramento (*training set*), generalmente molto corposo, è possibile dedurre una distribuzione di probabilità rappresentativa del dominio. La fase di validazione (*validation*) e la fase di test (*testing*) si occupano di valutare i risultati ottenuti dal modello su dati che non ha mai visto. Al contrario, effettuando previsioni su dati già visti dal modello, si otterrebbe

una valutazione troppo ottimistica delle prestazioni dell'algoritmo. Dopo la validation, in caso di performance scarse, si modificheranno i parametri del modello e verrà fatto ripartire il training finché il risultato sul validation set non sarà migliore. Il testing, invece, permette di misurare la capacità di generalizzazione del modello addestrato su nuovi dati.

Se il modello ottenuto è meno complesso della funzione che descrive il comportamento dei dati, come è possibile osservare in figura 1.1, si verifica *underfitting*. Ciò si verifica se si addestra un modello con pochi dati o si descrive la natura complessa dei dati con un modello troppo semplice: per esempio, usando un modello lineare per approssimare dati non lineari.

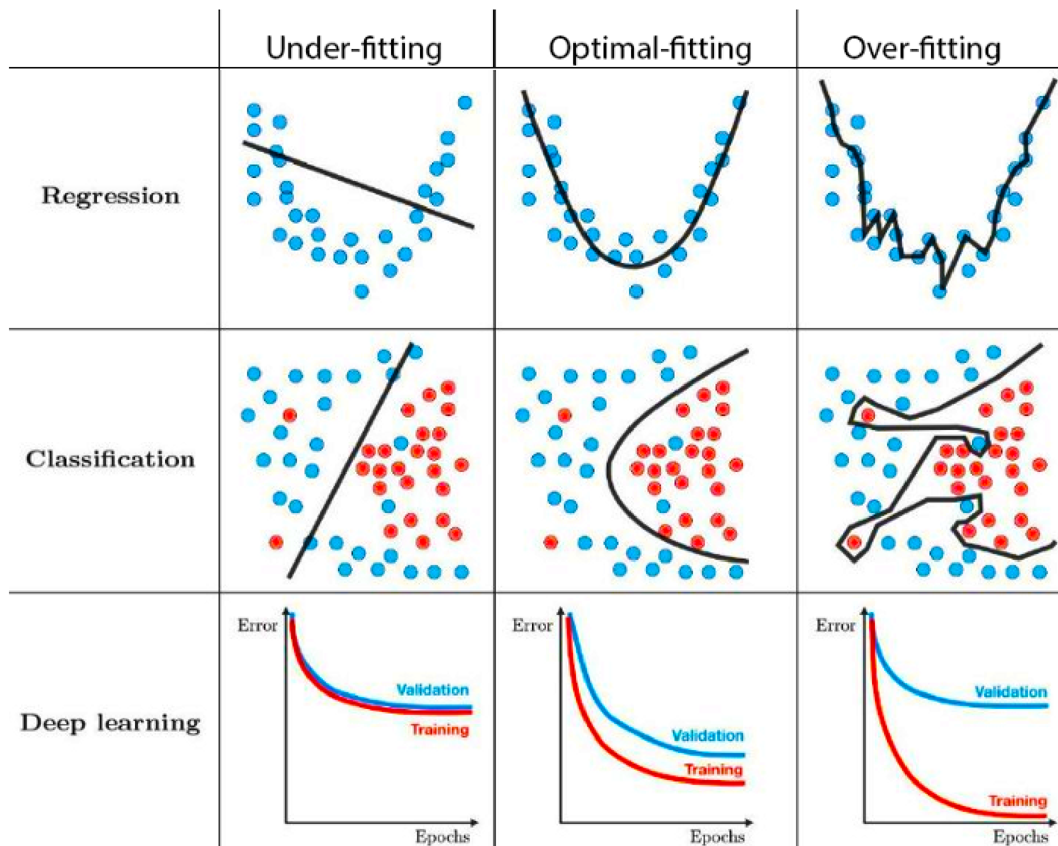


Figura 1.1: Rappresentazione del fitting di un modello.
Immagine tratta da towardsdatascience.com.

Se invece l'algoritmo risulta troppo complesso, potrebbe catturare rumore dai dati di training, cioè informazioni non rilevanti per il dominio, e causare *overfitting*, cioè non essere capace a generalizzare su nuovi dati. Ciò può essere evitato mediante uno dei seguenti accorgimenti:

- Addestrare il modello con più dati per evitare che si adatti troppo alle caratteristiche dei training examples (figura 1.2).

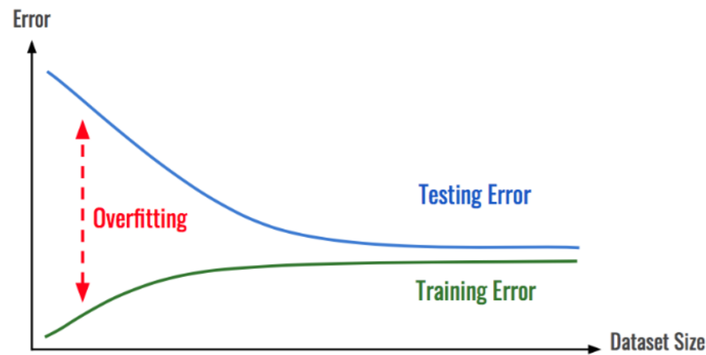


Figura 1.2: Sviluppo delle performance in base al numero di dati.
Immagine tratta da lorenzogovoni.com.

- Utilizzare la *K-fold cross validation*: metodo che permette di suddividere il dataset in sottoinsiemi e di effettuare su ognuno la validation in modo da valutare l'errore del modello su dati diversi e aggiustarlo di conseguenza.
- Rimuovere le feature non rilevanti.
- Eseguire un arresto anticipato (*early stopping*) durante l'addestramento prima che il modello perda la capacità di generalizzare. Il punto di interruzione viene definito durante la fase di validation (figura 1.3).

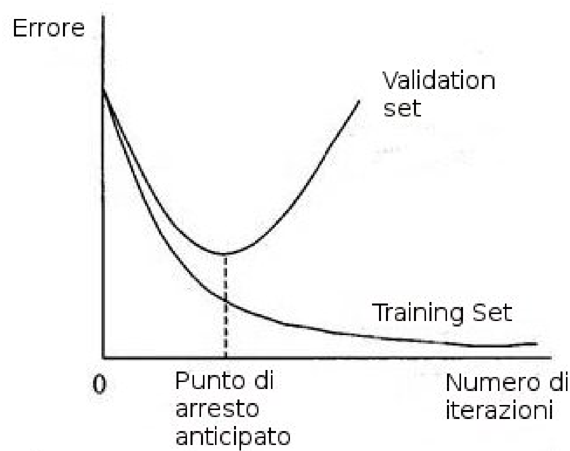


Figura 1.3: Punto di arresto anticipato per evitare overfitting.
Immagine tratta da lorenzogovoni.com.

- Applicare la regolarizzazione: metodo che semplifica il modello attraverso la minimizzazione dei valori dei pesi. Infatti, da quest'ultimi dipende il termine che viene aggiunto alla funzione di errore che l'addestramento ha l'obiettivo di minimizzare. In questo modo, i pesi non possono crescere esponenzialmente, rendendo il modello più generale.

Per estrarre un modello di conoscenza esistono diversi paradigmi:

- **Apprendimento supervisionato - Supervised Learning**

  Il modello viene addestrato su un dataset etichettato, cioè un insieme di input a cui viene associato il rispettivo output. L'obiettivo è trovare la funzione che minimizza l'errore tra le previsioni del modello e gli output reali. Tipicamente i modelli ottenuti con apprendimento supervisionato vengono applicati a due diverse categorie di problemi: la *classificazione*, dove gli input devono essere associati ad una predeterminata categoria di output, e la *regressione*, in cui l'output da predire è un valore numerico.

- **Apprendimento non supervisionato - Unsupervised Learning**

  Il modello viene addestrato su un insieme di dati non etichettati: l'algoritmo di apprendimento individua in modo autonomo le correlazioni comuni tra i dati. Questa tecnica permette di risparmiare risorse in termini di tempo, poiché non è più necessario effettuare l'etichettatura dei dati di addestramento. L'apprendimento non supervisionato viene generalmente utilizzato nei problemi di clustering, dove è necessario classificare un insieme di dati senza conoscere a priori le categorie di appartenenza: adottare un approccio automatico permette di estrapolare peculiarità e correlazioni che l'intelligenza umana può difficilmente cogliere.

- **Apprendimento per rinforzo - Reinforcement Learning**

  Questa tipologia di apprendimento consente all'algoritmo, definito in questo caso *agente*, di estrarre conoscenza senza sfruttare un insieme di dati in input, ma interagendo in modo dinamico con un ambiente, ricevendo un feedback per ogni azione compiuta. Il modello ha così l'obiettivo di massimizzare i feedback positivi ed eventualmente di adattare i comportamenti se riceve feedback negativi. Questo paradigma di apprendimento viene utilizzato, per esempio, per addestrare un modello a giocare ai videogiochi, o per realizzare sistemi di problem solving.

1.2 Rete Neurale

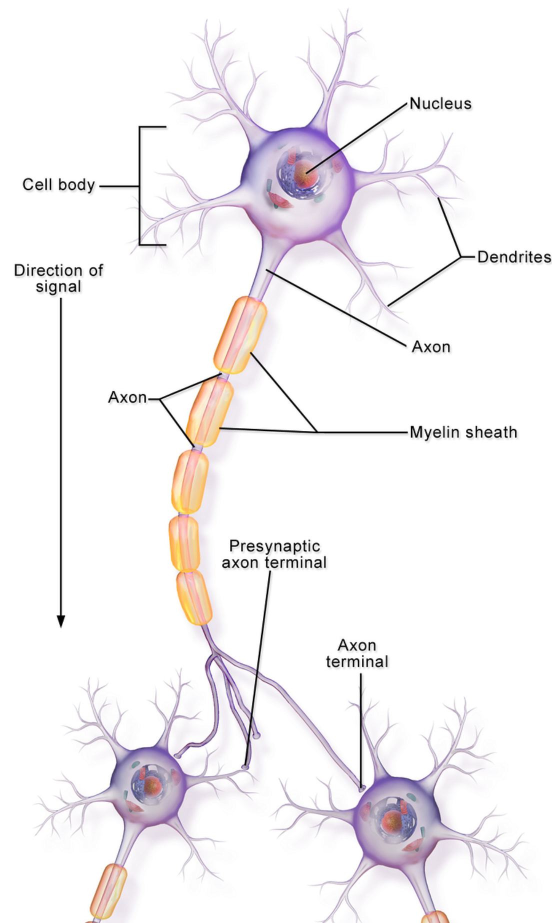


Figura 1.4: Struttura di un neurone biologico.
Immagine tratta da sawakinome.com.

La rete neurale è un sistema di apprendimento automatico ideato per simulare il funzionamento del cervello umano. Quest'ultimo contiene miliardi di neuroni (figura 1.4) che, a loro volta, possiedono ognuno migliaia di connessioni sinaptiche, che permettono il passaggio di informazioni tra i neuroni. Gli input vengono propagati al neurone attraverso i dendriti, collegati a loro volta alle sinapsi. I segnali in output attraversano gli assoni per raggiungere gli altri neuroni. Le cellule neuronali si formano prima della nascita insieme alle prime connessioni sinaptiche. Dopo la nascita, la creazione di nuovi neuroni subisce un arresto, ma l'apprendimento permette la formazione di nuove sinapsi.

1.2.1 Storia delle reti neurali

Il modello iniziale di *neurone artificiale* (figura 1.5) è stato creato dai ricercatori McCulloch e Pitts nel 1943 [1] ed era in grado di calcolare semplici funzioni booleane. La funzione g riceve gli input, esegue una sommatoria e in base al risultato la funzione f prende una decisione.

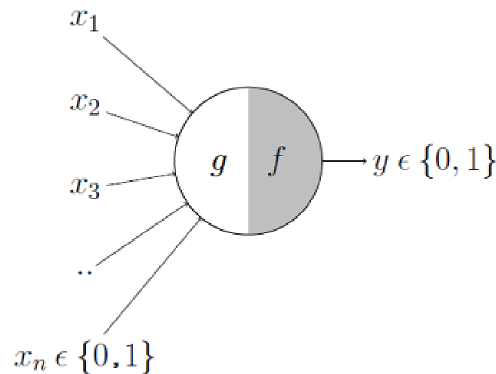


Figura 1.5: Modello iniziale di neurone artificiale.
Immagine tratta da researchdatapod.com.

Nel 1958, lo psicologo statunitense Frank Rosenblatt introduce il primo schema di rete neurale, detto *Percettrone* [2], per il riconoscimento e la classificazione di forme. Il percettrone è un tipo di classificatore binario, che mappa il vettore di input x in un valore di output *f(x)*, calcolato come segue:

$$f(x) = \chi(\langle w, x \rangle + b)$$

L'output dipende quindi dal vettore dei pesi w che, attraverso la sua modifica, ha lo scopo di apprendere e memorizzare i pattern dei dati in input: un notevole progresso rispetto al modello binario di McCulloch e Pitts in cui i pesi sono statici. Durante il decennio 1970-1980, le reti neurali ebbero un calo di popolarità e i finanziamenti alle ricerche subirono un arresto a seguito della pubblicazione di Marvin Minsky e Seymour A. Papert nel 1969 [3]. In essa vennero mostrati i limiti operativi delle reti a due strati basate sul percettrone, dimostrando l'impossibilità di risolvere tutti quei problemi non caratterizzati da soluzioni separabili linearmente. Nel 1974, si parla per la prima volta di *Multi-Layers Perceptron* (percettrone multistrato), soluzione che supera i limiti della versione senza strati intermedi. Nel 1986, venne introdotto l'algoritmo di *retropropagazione dell'errore* (error backpropagation), uno dei metodi più efficaci per l'addestramento delle reti neurali, che permette di aggiornare i pesi del modello al fine di ridurre l'errore tra le previsioni e i valori reali di output.

1.2.2 Reti neurali odierne

Le reti neurali odierne sono organizzate in layers (figura 1.6): vi è sempre un layer di input, uno di output e uno o più layers nascosti (hidden layers).

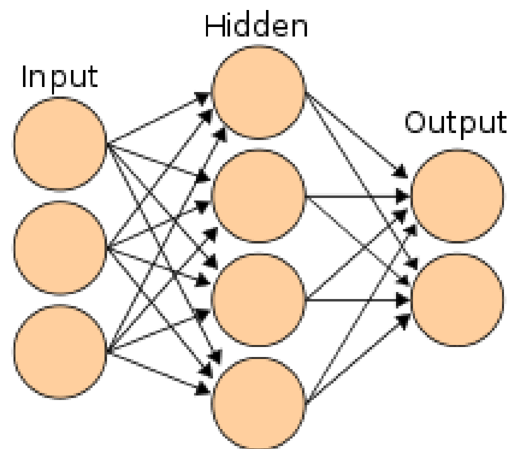


Figura 1.6: Struttura di una rete neurale semplice.
Immagine tratta da wikipedia.org.

Un neurone artificiale, come quello rappresentato in figura 1.7, riceve in ingresso i dati oltre ai rispettivi pesi. Quest'ultimi sono coefficienti associati ai singoli valori in input che ne determinano l'importanza.

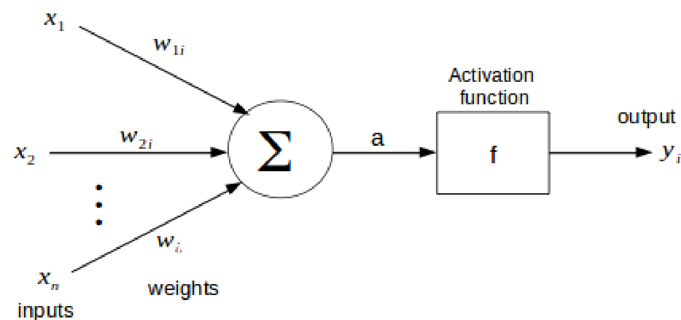


Figura 1.7: Struttura di un neurone artificiale.
Immagine tratta da abitofscience.altervista.org.

La funzione di attivazione f è continua e determina l'output del neurone in funzione della somma pesata degli input. Per esempio, esistono la funzione sigmoide, la ReLU o la tangente iperbolica.

L'addestramento di una rete neurale consiste nel determinare il valore dei pesi che stabiliscono il mapping tra input e output in base al principio della

discesa del gradiente. Quest'ultimo permette di valutare gli errori del modello e di correggerne i parametri con la backpropagation, minimizzando la *funzione di errore* fino a raggiungere un livello di accuratezza ottimale, in genere con una percentuale di confidenza fissata al 95%.

Nelle reti *feed-forward*, i dati percorrono la rete da sinistra verso destra (dall'input all'output), ma esistono anche reti più complesse come le Recurrent Neural Network (RNNs) e le Convolutional Neural Network (CNNs).

1.3 Deep Learning

Il Deep Learning (figura 1.8) è una tecnica di apprendimento automatico che sfrutta le reti neurali profonde, cioè reti composte da due o più hidden layers. I layers sono disposti gerarchicamente e permettono di gestire progressivamente la complessità dei problemi: quelli più bassi (più vicini all'input) riescono a riconoscere i pattern più semplici, mentre quelli più alti possono individuare i pattern più complessi. Nel Machine Learning il programmatore effettua manualmente la selezione delle feature, cioè la scelta delle variabili rilevanti per la risoluzione del problema. Diversamente, gli algoritmi di Deep Learning effettuano la selezione autonomamente, ma per permettere ciò la rete neurale deve analizzare un'enorme quantità di dati.

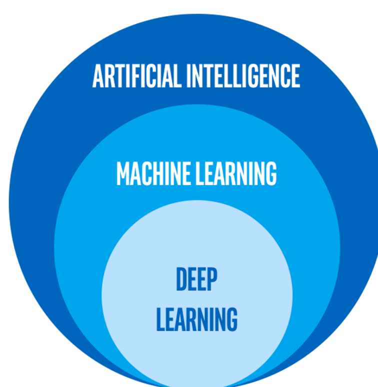


Figura 1.8: Gerarchia delle discipline di Intelligenza Artificiale.
Immagine tratta da intel.it.

Le Deep Neural Network si ispirano alla complessità del cervello umano, che sfrutta diverse aree dello stesso: ognuna di queste prende in ingresso informazioni dalle aree precedenti e le processa per restituirle a quelle successive. Il numero di livelli determina così la complessità della rete neurale, ma anche la bontà del risultato, in quanto ogni strato aggiunge informazioni all'output.

1.4 Transfer Learning

Come detto in precedenza, l'addestramento di una rete neurale richiede un'enorme quantità di dati e giorni di addestramento. Disporre di dati sufficienti può risultare complicato e costoso, così è stata introdotto il Transfer Learning, una nuova tecnica di Machine Learning.

Attraverso questa tecnica, è possibile riutilizzare una rete pre-addestrata su un task simile a quello per cui è stata addestrata la rete. Questo tipo di modelli necessitano di essere abbastanza generali per essere efficaci sul nuovo task. È possibile reimpiegare un modello pre-addestrato attraverso il *fine-tuning*, dove gli ultimi strati della rete pre-addestrata vengono rimossi, o aggiunti nuovi, e il modello viene ri-addestrato per ottimizzare i pesi rispetto al nuovo task. In questo modo, il tempo per addestrare i modelli si riduce drasticamente, mantenendo allo stesso tempo alti livelli di accuratezza.

Capitolo 2

Natural Language Processing

2.1 Cos'è il NLP?

Il Natural Language Processing (NLP) è una disciplina del campo dell'intelligenza artificiale che ha l'obiettivo di progettare sistemi in grado di processare, interpretare o generare testo in linguaggio naturale, aspirando a raggiungere performance e abilità paragonabili a quelle di un essere umano.

Per linguaggio naturale si intende la lingua parlata e scritta dall'uomo, che può articolarsi secondo i diversi idiomi e quindi seguire regole linguistiche differenti. L'elaborazione del linguaggio naturale presenta diverse difficoltà dovute ad ambiguità nel significato dei termini, elementi non previsti dalla lingua, come abbreviazioni ed emoticon, complessità sintattica e forme linguistiche complesse da interpretare, come il sarcasmo, l'ironia e le figure retoriche.

I sistemi di elaborazione del linguaggio naturale hanno avuto origine nel 1950 quando Alan Turing pubblicò un articolo intitolato "*Computing Machinery and Intelligence*" [4], introducendo il conosciuto *Turing test*, ossia un criterio per stabilire se una macchina possa essere definita intelligente, in grado quindi di esprimersi in modo significativo e indistinguibile da un essere umano.

Per tre decenni, dal 1950 ai primi anni '90, i sistemi di NLP seguirono un approccio *rule-based*: ogni algoritmo basava il proprio funzionamento su un insieme di regole predefinite dal programmatore (hand-written rules), generalmente redatte singolarmente per ogni lingua, in quanto differenti tra loro. La definizione di questi algoritmi comportava molte ore di lavoro e perciò alla fine degli anni '80 prese piede un approccio definito *statistical-based*.

Lo Statistical NLP introdusse l'utilizzo del Machine Learning, con l'obiettivo di estrarre regole linguistiche dal testo automaticamente. Per garantire il funzionamento di questi algoritmi, i modelli vengono addestrati mediante l'utilizzo dei cosiddetti *corpora linguistici*, ossia grandi dataset di testo etichettati.

Oggigiorno, i sistemi di Deep Learning possiedono i migliori risultati allo stato dell'arte in ambito NLP.

2.2 Tecniche di pre-processing

Il testo è un tipo di *dato destrutturato*, cioè un dato che viene conservato senza alcuno schema, per cui non è possibile estrapolarne direttamente caratteristiche e informazioni. Per essere utilizzato nei sistemi di NLP, al testo vengono applicate le cosiddette *tecniche di pre-processing*, che estraggono da esso elementi di base strutturati e significativi. Tra queste tecniche vi sono:

- *Tokenizzazione*: il testo viene segmentato in elementi sintattici di base, come parole (word tokenization) o frasi (sentence tokenization).

- *Part of Speech Tagging (POS)*: una volta effettuata la segmentazione del testo a livello di parole, è possibile etichettare ogni elemento ottenuto con un'etichetta che ne indica la classe grammaticale (POS), ossia il ruolo che svolge nella preposizione (nome, verbo, aggettivo, etc.)

Le tecniche descritte successivamente vengono introdotte per applicare il cosiddetto *text cleaning*, cioè "pulire" il testo da informazioni meno rilevanti, come filtrare sequenze di parole, rimuovere elementi non necessari e uniformare i termini polisemici, cioè con lo stesso significato.

- *Casefolding*: il testo viene convertito tutto in minuscolo o maiuscolo.

- *Rimozione delle stopword*: vengono rimosse parole che non danno informazioni sulla semantica (significato) della frase, come articoli, preposizioni, congiunzioni e verbi ausiliari. La selezione delle parole avviene in base a una lista predefinita, chiamata stoplist, che varia per ogni lingua.

- *Lemmatizzazione*: ogni parola viene sostituita con il suo lemma, cioè la forma base che si trova nel dizionario linguistico, in modo da uniformare i termini simili e ridurre il numero di parole univoche nel vocabolario.

- *Stemming*: da ogni parola viene estratta la rispettiva radice morfologica, ossia l'elemento irriducibile che esprime il significato della stessa. A volta la radice può non avere un significato compiuto e questa tecnica potrebbe comportare la perdita di informazioni: nonostante l'implementazione più semplice, questo metodo può restituire risultati non soddisfacenti.

2.3 Rappresentazione del testo

Le reti neurali non sono in grado di elaborare le stringhe testuali, ma possono processare solo dati numerici: vi è quindi la necessità di convertire il testo in un formato idoneo. Esistono diverse tecniche per la rappresentazione del linguaggio che permettono di codificare le parole.

2.3.1 One-hot Encoding

Un primo metodo, semplice e poco performante, è chiamato ***One-hot Encoding*** e consiste nel creare un vettore di 0 e 1 per ogni parola e simbolo in input. Un *one-hot vector* identifica univocamente una parola, quindi non possono esservi due rappresentazioni uguali per parole diverse (sono considerate a sé stanti anche due parole ortograficamente identiche ma posizionate in punti diversi della sequenza). Tale rappresentazione permette di codificare la posizione della parola all'interno del testo e risulta molto utile quando viene applicata a problemi di classificazione o di regressione.

2.3.2 Modello n-gram

Il ***modello n-gram*** permette di suddividere la sequenza di testo in sottoinsiemi di parole di lunghezza definita dall'ampiezza della finestra. La finestra scorre, una parola per volta, lungo la sequenza e raggruppa le n parole consecutive che trova. Un modello *bi-gram* può raggruppare due parole per volta, mentre un modello *tri-gram* ne considera tre ad ogni step. Le sequenze di parole estrapolate possono evidenziare correlazioni significative tra i termini o non essere di particolare utilità in quanto codificate a priori. Il modello n-gram è anche definibile come un modello probabilistico in grado di predire il prossimo elemento a partire da una sequenza (di lunghezza 1 nei modelli bi-gram e 2 nei modelli tri-gram) sulla base di quelle codificate. É quindi molto utile in applicazioni di generazione del testo, auto-completamento e NLP in generale.

2.3.3 Bag of words

La rappresentazione tramite il modello ***bag of words*** restituisce un multiset delle parole contenute in un documento: per ognuna di queste viene indicata la frequenza con cui compare nella sequenza di testo. Tale metodo è spesso utilizzato per classificare il contenuto del testo: per esempio, per definire il sentiment di una recensione, mediante sentiment analysis, o per implementare filtri di spam sulla base di un set predefinito di parole. Può anche essere sfruttato in applicazioni di visione artificiale come l'object recognition, dove

ogni caratteristica estrapolata dall'immagine è equivalente ad una parola nel multi-set. Spesso il numero di occorrenze di una parola nel testo non è indicativo della sua importanza, in quanto quel termine potrebbe non contribuire a definire la semantica del documento. Perciò vengono applicati degli schemi, come il *TF-IDF*, per normalizzare tali valori. Tale schema misura il peso di un termine in base ad un fattore locale *TF* (term frequency), ossia il numero di occorrenze all'interno di un certo documento, ed un fattore globale *IDF* (inverse document frequency), che pesa l'importanza del termine in una collezione di documenti: se il termine compare in pochi documenti allora ha più carattere distintivo.

2.3.4 Word embedding

Esistono altre tecniche che codificano univocamente le parole, mantenendo anche informazioni riguardo alla sintassi e alla semantica della frase. Queste sono raggruppate sotto il termine ***word embedding*** e permettono di proiettare le rappresentazioni delle parole, sotto forma di vettori, in uno spazio vettoriale. La posizione e le proporzioni dei vettori sono indice delle relazioni (similarità, sovrapposizione, differenza, etc.) che intercorrono tra i termini e che possono essere estrapolate effettuando delle operazioni matematiche (figura 2.1).

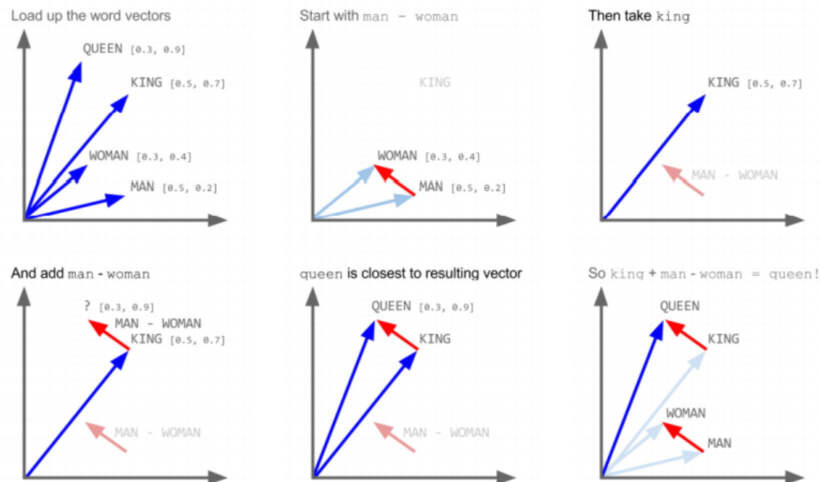


Figura 2.1: Esempio di estrazione delle relazioni tra diversi termini.
Immagine tratta da multithreaded.stitchfix.com.

Tra gli algoritmi più utilizzati vi sono *Word2Vec* [5] e *GloVe* [6]. Con la pubblicazione "*Distributed Representations of Sentences and Documents*" [7], è stato introdotto un algoritmo in grado di generare rappresentazioni di sequenze arbitrariamente lunghe come frasi, paragrafi o documenti, fedeli alla struttura sintattica e semantica del testo. Da questo studio deriva l'algoritmo *Doc2Vec*.

Un vantaggio degli algoritmi “word embedding” è che sono di tipo semi-supervisionato, cioè non necessitano di dataset etichettati, ma sfruttano gli stessi dati non etichettati per creare etichette utili all’addestramento del modello.

2.4 Applicazioni

Il Natural Language Processing è un campo di studio strategico, in quanto i dati testuali sono una fonte di informazioni incommensurabile. Internet, i sistemi informatici o quelli analogici forniscono un’enorme quantità di dati, e nel tempo si sono sviluppati numerosi ambiti applicativi:

- *Sentiment analysis*: è una tecnica che estrae opinioni dal testo e classifica se sono positive, negative o neutre. Risulta molto utile per interpretare il sentiment degli utenti rispetto un servizio o un prodotto.
- *Question answering*: riguarda lo sviluppo di applicazioni in grado di generare una risposta in seguito a una domanda.
- *Chatbot e assistenti vocali*: sistemi ottimizzati per la comprensione del linguaggio naturale, scritto o parlato, e la gestione di dialoghi con l’utente.
- *Machine translation*: permette di generare automaticamente traduzioni tra lingue diverse, un esempio noto è Google Translate.
- *Automatic summarization*: vasta serie di sistemi in grado di sintetizzare automaticamente un insieme di dati come il testo, immagini o video.
- *Syntactic analysis*: analizza la struttura di una sequenza in input e ne determina la correttezza in base ad un insieme di regole grammaticali.
- *Topic analysis*: estrae da uno o più documenti gli argomenti chiave.

2.5 Tecnologie di Deep Learning

Negli ultimi anni, le tecnologie di Deep Learning hanno permesso di migliorare sensibilmente lo stato dell’arte in ambito NLP e di affrontare problemi per cui non si era trovata ancora una soluzione soddisfacente. Rispetto ai sistemi precedenti, quelli basati sulle reti neurali profonde hanno la capacità di apprendere in modo automatico dai dati e di migliorare le proprie prestazioni durante l’addestramento grazie agli esempi che gli vengono presentati.

2.5.1 Recurrent Neural Network

Le *Recurrent Neural Network* (RNN) sono una classe di reti neurali che, a differenza delle reti feed-forward, hanno anche punti di connessione all'indietro: per ciascun step, ogni neurone riceve sia il vettore di input che il vettore di output del precedente time-step (figura 2.2).

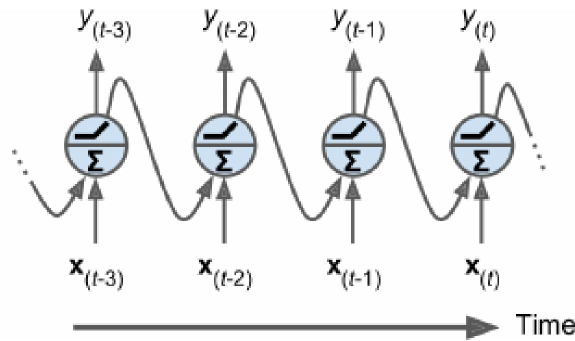


Figura 2.2: Neurone di una rete RNN attraverso il tempo.
Immagine tratta da [8].

In questo modo, ogni neurone ha una sorta di memoria che sfrutta per imparare pattern della sequenza di input, per ciò viene definito *cella di memoria*. Queste reti sono ideali per processare sequenze di dati di lunghezza arbitraria, come il testo. Sono definite reti *sequence-to-sequence*, in quanto elaborano sequenze in input e ne producono in output, anche simultaneamente.

Un limite di questi sistemi è la "perdita di memoria" per input lunghi. Infatti, per processare sequenze di lunghezza maggiore, una RNN dovrebbe essere addestrata per numerosi time-step attraverso una deep neural network: questo causa (i) instabilità dei gradienti, cioè il problema di aggiornare i pesi aumentandoli fino a farli esplodere, e (ii) la possibilità di dimenticare i pattern appresi per i primi input della sequenza, in quanto i dati subiscono numerose trasformazioni durante l'attraversamento della rete.

La normalizzazione è un metodo che permette di distribuire i dati in input su una scala uniforme ovviando all'instabilità dei gradienti. La tecnica di *Batch Normalization* prende in considerazione gli input di un singolo neurone in un batch (insieme raggruppato di input che viene processato simultaneamente). Il *Layer Normalization*, invece, considera l'insieme degli input in un determinato layer, risultando più efficiente nelle reti ricorrenti.

Per ovviare invece alla perdita di memoria di queste reti, sono state sviluppare celle di memoria a lungo termine:

- *Long Short-Term Memory (LSTM)*: lo stato della cella viene suddiviso in short-term state, h(t), e long-term state, c(t). All'interno della cella (figura

2.3), lo stato passa attraverso i *gate controller*, grazie ai quali il neurone impara a riconoscere gli input importanti (input gate), memorizzandoli in un long-term state e preservandoli finché necessari (forget gate).

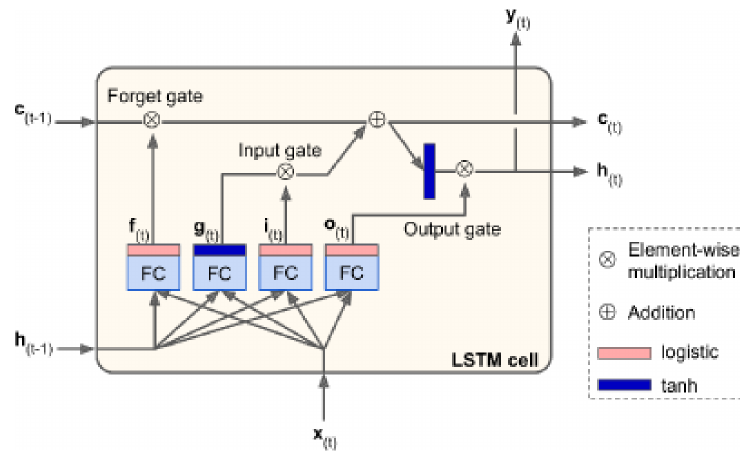


Figura 2.3: Struttura di una cella LSTM.
Immagine tratta da [8].

- *Gated Recurrent Unit (GRU)*: sono una versione semplificata delle celle LSTM in cui i vettori di stato sono uniti nel singolo vettore h(t). Il gate controller z(t) funge da forget gate e input gate, e il gate controller r(t) decide quale parte del precedente stato verrà mostrata al main layer g(t).

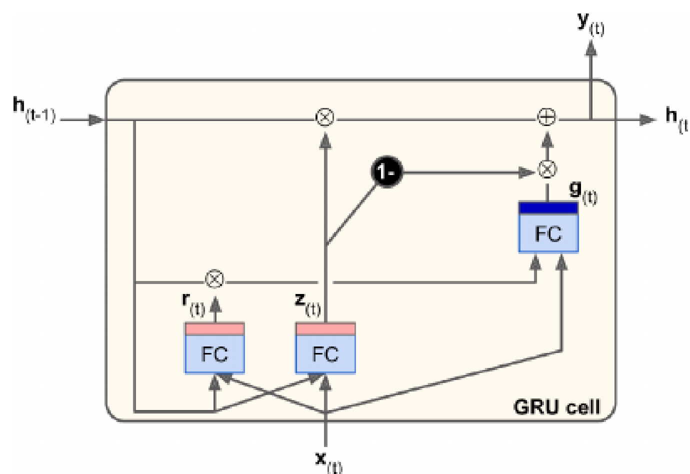


Figura 2.4: Struttura di una cella GRU.
Immagine tratta da [8].

2.5.2 Convolutional Neural Network

Le Convolutional Neural Network (CNN) sono reti neurali profonde la cui struttura è ispirata a quella della corteccia visiva dell'essere umano. Come in quest'ultima, una rete CNN è costituita da più strati, ognuno con un compito diverso. La presenza dei livelli di convoluzione permette, attraverso l'utilizzo dei filtri, di estrarre caratteristiche nei dati. Inoltre, se vengono impilati più layers convoluzionali, è possibile individuare features a diversi livelli di astrazione. I dati che passano attraverso la rete sono matrici (figura 2.5), che possono rappresentare immagini, per esempio nei task di visione artificiale, o frasi e documenti, nel caso di sequenze testuali.

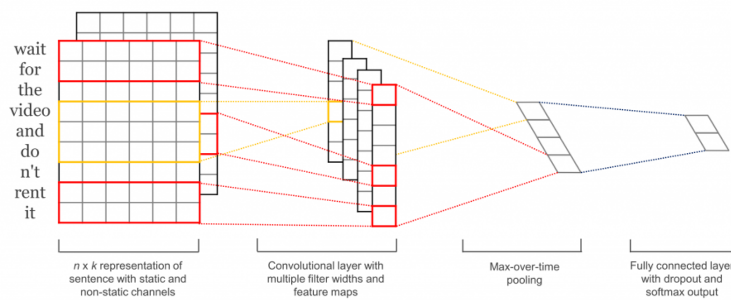


Figura 2.5: Convolutional Neural Network.
Immagine tratta da wildml.com.

Durante l'addestramento, il modello aggiorna il valore dei filtri nei livelli di convoluzione, ma questi sono solitamente seguiti dalle funzioni di pooling, che permettono di ridurre la dimensione delle matrici. É possibile utilizzare un layer convoluzionale monodimensionale per processare sequenze molto lunghe: questo fa scorrere diversi filtri lungo una sequenza, producendo una mappa monodimensionale di feature. Ogni filtro impara a rilevare un singolo pattern sequenziale molto corto: con 10 filtri, il risultato del layer sarà composto da 10 sequenze monodimensionali, equivalenti a una sequenza in 10 dimensioni.

L'architettura WaveNet (figura 2.6) permette di impilare layer convoluzionali a una dimensione, raddoppiando il dilation rate, cioè la misura della distanza degli input di ciascun neurone, permettendo ai layers più bassi di imparare short-term patterns e ai layers più alti di imparare long-term pattern.

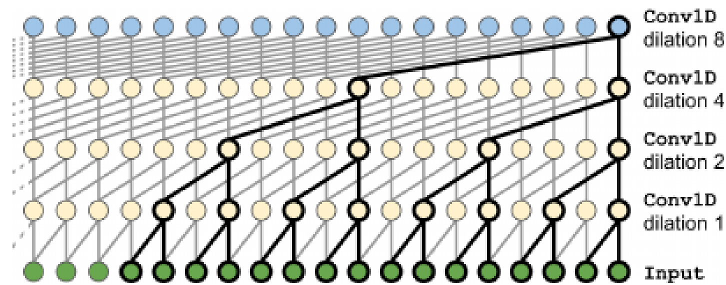


Figura 2.6: Architettura WaveNet.
Immagine tratta da [8].

2.5.3 Encoder-Decoder

Vi sono altre tipologie di reti in grado di processare sequenze di dati:

- *Sequence-to vector network*: prende una sequenza di input e ignora tutti gli output tranne l'ultimo. Viene anche definito *Encoder*.
- *Vector-to-sequence network*: prende un vettore in input e lo sottopone alla rete ad ogni time step e restituisce una sequenza in output. Viene anche definito *Decoder*.

Un modello *encoder-decoder* mette in sequenza queste due tipologie di reti ed è in grado di processare in input una sequenza e di generarne una in output, spesso di lunghezza diversa dalla prima. Per questo viene anche definito *sequence-to-sequence model* (Seq2Seq).

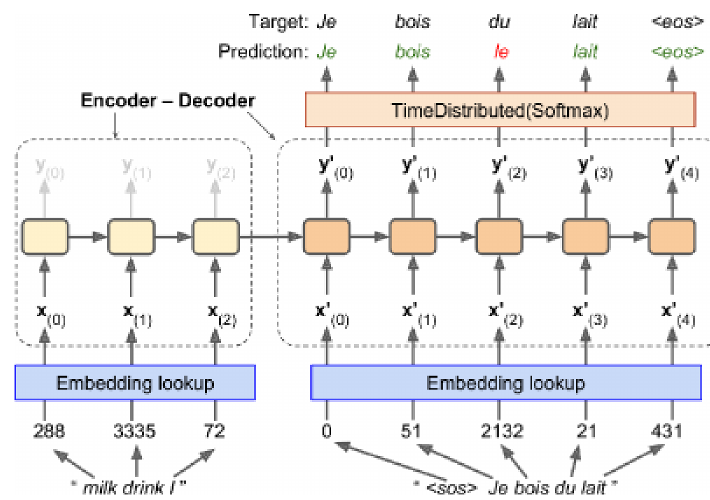


Figura 2.7: Modello encoder-decoder applicato alla traduzione.
Immagine tratta da [8].

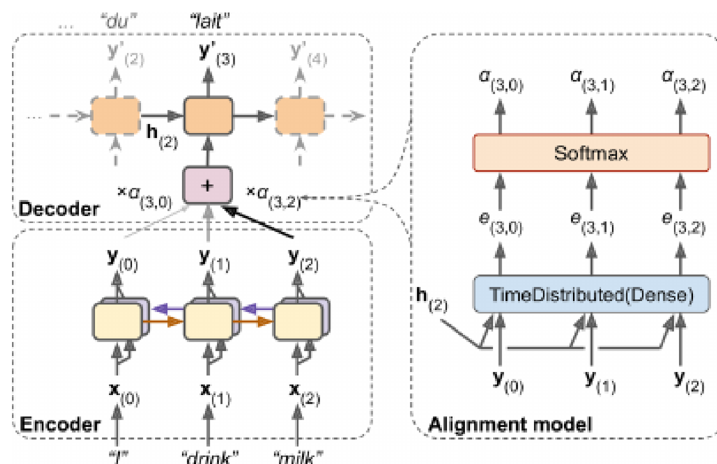


Figura 2.8: Modello encoder-decoder con Concatenative Attention.
Immagine tratta da [8].

In figura 2.7, è possibile notare che le frasi in inglese vengono affidate all'encoder e che il decoder restituisce le traduzioni in francese. La frase "*I drink milk*" viene ribaltata prima di essere passata all'encoder in modo che ogni parola venga processata prima dal decoder. Successivamente, il decoder restituisce un punteggio per ogni parola nel vocabolario di output (francese), ed infine il layer di softmax riformula questi punteggi come probabilità. Le parole con un valore alto di probabilità saranno l'output finale.

Le prestazioni di un modello encoder-decoder potrebbero non essere ottimali, in quanto la rete non è in grado di tornare indietro e correggere i propri errori. Le tecniche attuali mettono a disposizione un metodo che migliora le predizioni di modelli: il *beam search*. Quest'ultima tiene traccia delle k frasi più promettenti: ad ogni passo il decoder aggiunge una parola alle frasi selezionate e le probabilità vengono aggiornate ottenendone delle nuove, condizionate dalle precedenti. L'iter viene ripetuto finché non si ottiene una traduzione ottimale.

2.5.4 Attention Mechanism

Il modello encoder-decoder non è in grado di processare sequenze di testo molto lunghe, in quanto possiede una memoria "a breve termine", cioè non può conservare per molti time step le rappresentazioni delle parole.

Per ovviare a ciò, Dzmitry Bahdanau introdusse la tecnica *Concatenative Attention* (o Bahdanau attention), che permette al decoder di focalizzarsi sulle parole appropriate ad ogni time step. Questa tecnica prevede che il decoder riceva tutti gli output dell'encoder, non solo lo stato finale (figura 2.8). In questo modo, ad ogni time step, la *memory cell* (in colore rosa) del decoder

può calcolare una somma pesata di tutti gli output e determinare su quali il decoder dovrà porre la sua attenzione in quel momento. I pesi vengono calcolati dall'*Alignment model* (o attention layer) e indicano il grado di similarità tra gli output dell'encoder e lo stato del decoder al precedente time step. All'interno di questo vi è il *TimeDistributed Dense layer* che si occupa di calcolare i punteggi; la funzione di softmax li trasforma in valori di probabilità.

La versione *Multiplicative Attention* (o Lounge Attention) prevede di misurare il grado di similarità attraverso il prodotto scalare dei rispettivi vettori. L'approccio *General Dot Product* prevede una trasformazione lineare degli output dell'encoder, prima che venga effettuato il prodotto scalare.

Dal punto di vista matematico, il meccanismo di attenzione si basa sui seguenti passaggi:

1. Creazione delle matrici *queries* (Q), *keys* (K) e *values* (V), a partire da una determinata sequenza (figura 2.9).

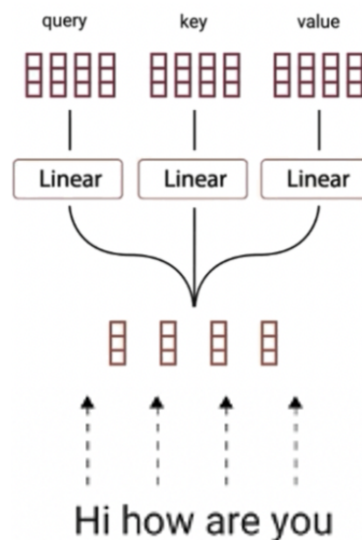


Figura 2.9: Creazione delle matrici *queries* (Q), *keys* (K) e *values* (V) nel meccanismo di attenzione.

2. Calcolo della matrice *scores*, determinata dal prodotto delle matrici Q e K: il risultato determina quanto le parole devono "prestare attenzione" alle altre, sulla base della loro importanza per esprimere il concetto stesso della parola e della frase.

3. Applicazione della funzione di *softmax* alla matrice *scores*, in modo da ottenere dei valori di probabilità (tra 0 e 1), definiti *attention weights*.

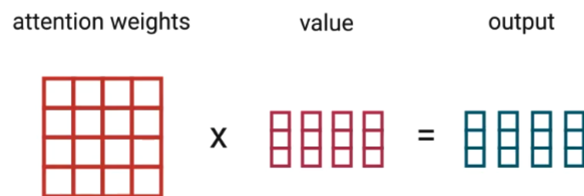


Figura 2.10: Calcolo degli output nel meccanismo di attenzione.

4. Prodotto tra la matrice *attention weights* e la matrice *values*, per ottenere una matrice di *output* che rappresenterà i token della sequenza su cui prestare attenzione (figura 2.10).

Il meccanismo di attenzione (*Attention Mechanism*) non viene utilizzato solo in ambito NLP, ma può essere applicato anche in task di visione artificiale: per esempio, permette al decoder di focalizzarsi su una parte dell'immagine in input per generare una didascalia appropriata (image captioning).

Oltre a migliorare le performance dei modelli encoder-decoder, il meccanismo di attention apporta benefici alla comprensione del modello (*explainability*), in quanto rende più chiara la ragione di un determinato risultato e facilita la correzione di previsioni sbagliate.

2.5.5 Transformer

Il *Transformer* è un modello di deep learning encoder-decoder che sfrutta l'*Attention Mechanism*, proposto per la prima volta da un team di ricerca di Google [9]. É ideale per processare sequenze di dati in input, come il testo, e viene applicato in numerosi ambiti: machine translation, text summarization, text generation, sequence analysis e video understanding.

L'architettura Transformer ha migliorato significativamente lo stato dell'arte in ambito NLP: in precedenza, i sistemi LSTM e GRU arricchiti dal meccanismo di attenzione detenevano i migliori successi in questo campo. Non viene sfruttato nessun layer ricorrente (RNN) e non è richiesto che gli elementi della sequenza vengano processati in ordine, perciò il modello diventa parallelizzabile e i tempi di training si riducono notevolmente. Grazie a questi vantaggi, si sono sviluppati numerosi modelli pre-addestrati su grandi corpora.

In figura 2.11, a sinistra, è rappresentato l'encoder, che prende in input una sequenza di parole codificate come word embedding dall'input embedding. A destra vi è il decoder, che riceve una sequenza di input shiftata a destra di una posizione, anch'essa codificata come sequenza di word embedding. Ad ogni word embedding viene aggiunto il *positional embedding*, ossia un vettore che codifica la posizione di ogni parola all'interno della frase.

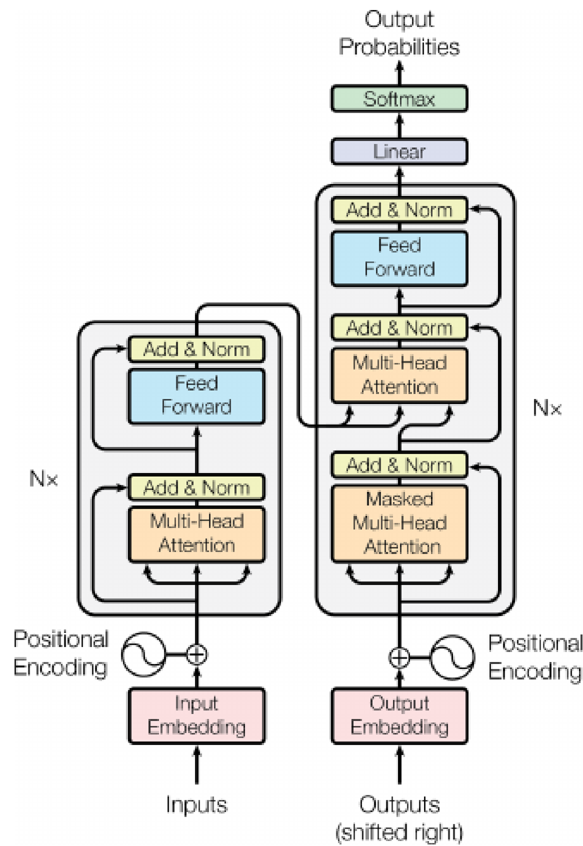


Figura 2.11: Architettura del modello Transformer.
Immagine tratta da [9].

Il *Multi-Head Attention layer* determina le relazioni che intercorrono tra ogni parola e il resto della frase (self-attention mechanism), dando maggior peso a quelle più rilevanti. Il primo step (figura 2.12) applica una serie di trasformazioni lineari a Q, K e V, proiettandoli in sottospazi differenti, ciascuno focalizzato su determinate caratteristiche della parola.

Il layer Scaled Dot-Product (figura 2.13) implementa la fase di lookup tra le query e le chiavi calcolando delle misure di similarità (MatMul) che verranno scalate (Scale) per evitare la saturazione della funzione di softmax, che si occuperà di convertirle in probabilità. Il layer Mask è utilizzato per ignorare determinate coppie chiave/valore nella funzione di softmax. Per esempio, viene utilizzato nel *Masked Multi-Head Attention layer* che, a differenza del Multi-Head Attention layer, si focalizza sulle parole posizionate precedentemente a quella corrente, in quanto il decoder ha accesso solo alle parole già restituite.

Infine, i risultati del meccanismo di attenzione vengono concatenati (Concat) e proiettati nello spazio originale (Linear).

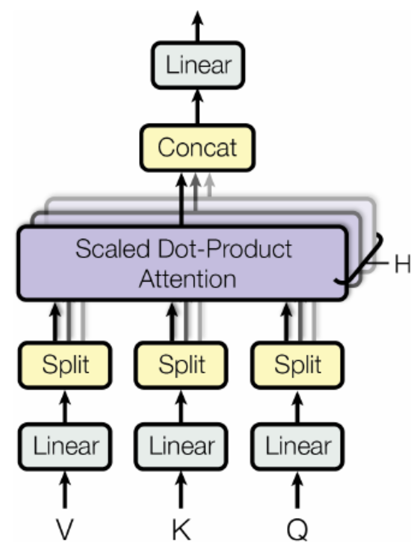


Figura 2.12: Architettura del Multi-Head Attention layer.
Immagine tratta da [9].

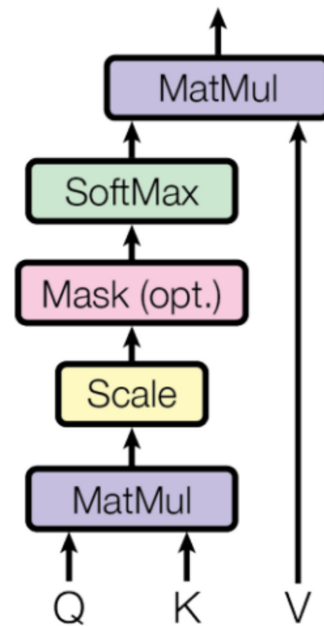


Figura 2.13: Architettura del Scaled Dot-Product Attention layer.
Immagine tratta da [9].

Capitolo 3

Automatic Text Summarization

Per questioni di tempo o praticità, quando si prende in mano un testo molto lungo, si vorrebbe avere la possibilità di leggerne un estratto o un riassunto per conoscere nell'immediato le informazioni più importanti. Produrre questo tipo di documenti richiede in certi casi molto tempo.

L'Intelligenza Artificiale mette a disposizione processi automatici che permettono di estrarre le informazioni più rilevanti da un insieme di dati in input, producendo un sottoinsieme in output. Questo gruppo di metodi viene definito come *Automatic Summarization* e possono essere applicati a immagini, video o testi (*Text Summarization*).

I metodi di Text Summarization permettono di ottenere un riassunto fluente e significativo a partire da un documento in input più lungo. Vi sono due principali tipologie di approccio:

- **Extractive Summarization**: il modello esegue un ranking delle frasi, seleziona le più importanti e informative, e infine le utilizza per creare il riassunto. Questo metodo risulta essere semplice, ma poco flessibile.

- **Abstractive Summarization**: questo approccio identifica le informazioni più rilevanti e il contesto per riformularle e sintetizzarle. Le frasi nel riassunto non sono estratte dal documento di origine ma vengono generate ex-novo: ciò permette di ottenere un risultato più vicino a quello *human-written* (scritto dall'uomo). I sistemi che adottano questo approccio potrebbero risultare svantaggios, in quanto più complessi, poiché necessitano di un componente aggiuntivo: il generatore.

In base al numero di documenti da riassumere, si parla di ***Single-document Summarization*** **(SDS)** o ***Multi-document Summarization*** **(MDS)**.

3.1 Extractive Summarization

Generalmente il riassunto scritto da una persona è una rielaborazione del testo, più o meno complessa, ma le tecniche di *extractive summarization* hanno ottenuto notevole successo, in quanto il compito di generare ex-novo un testo in modo automatico, come avviene nei sistemi di *abstractive summarization*, risulta essere più difficile.

Tutti i sistemi estrattivi seguono il seguente funzionamento [10]:

1. Viene costruita una **rappresentazione intermedia del testo** che definisce le informazioni più importanti.
2. Si assegnano dei **punteggi alle frasi** in base alla loro rappresentazione.
3. Il riassunto finale è composto dalle n frasi con punteggio più alto. La **selezione delle frasi** può avvenire nei seguenti modi:
    - Tramite algoritmi *greedy*.
    - Attraverso l'ottimizzazione di problemi che vogliono massimizzare la portata informativa del riassunto e minimizzarne la ridondanza.
    - In base al tipo di documento: per esempio, le pagine web contengono link ad altri siti, o gli articoli scientifici menzionano altre fonti, che possono contenere informazioni correlate e utili al riassunto finale.

Rappresentazione intermedia del testo

Vi sono due principali tipologie per la rappresentazione intermedia del testo:

1. **Topic Representation Approaches**

    Identificano le parole che meglio descrivono il tema del testo.

    - **Topic Words**

      Una delle prime tecniche, presentata negli anni '50 [11], definisce il *topic* del documento in base alle parole più frequenti. Una volta individuate quest'ultime (*topic words* o *topic signature*), l'importanza di una frase può essere calcolata in base alla frequenza assoluta o relativa con la quale queste ricorrono in essa.
    - **Frequency-driven Approaches**

      Si assegnano dei pesi alle parole nella rappresentazione intermedia del documento. Le tecniche più utilizzate sono il TF-IDF e la *Word Probability*, in cui si definisce il peso di una parola come il numero di occorrenze nel documento diviso il numero totale di parole.

- **Latent Semantic Analysis** [12]

  Si tratta di un metodo non-supervisionato per costruire la rappresentazione semantica di un testo. Viene inizialmente costruita una matrice *termini-frasi*, in cui ogni cella corrisponde al peso della parola nella frase, calcolato tramite TF-IDF. Successivamente, viene effettuata una trasformazione $A = U\Sigma V^T$ della matrice A, tramite *SVD (Singular Value Decomposition)*. La matrice U contiene i pesi delle parole rispetto ai topic, la matrice Σ i pesi dei topic e la matrice V^T i pesi delle frasi rispetto ai topic. La matrice $D = \Sigma V^T$ definisce quanto una frase rappresenta un topic: l'idea è di sceglierne una per ogni tema oppure di assegnare un termine di importanza ai topic per definire un numero di frasi da estrarre.

2. **Indicator Representation Approaches**

   La rappresentazione del testo viene definita con un insieme di features che vengono direttamente utilizzate per calcolare il ranking delle frasi.

   - **Graph Methods**

     In questo caso il documento viene rappresentato attraverso un grafo connesso: i vertici definiscono le frasi e gli archi indicano la similarità che intercorre tra esse. Un sotto-grafo può rappresentare un topic: le frasi più connesse all'interno di un sottoinsieme saranno quelle che, con maggior probabilità, verranno incluse nel riassunto.

   - **Machine Learning**

     La probabilità con cui una frase può essere inclusa nel riassunto viene calcolata con tecniche di machine learning. Solitamente, si considera diverse features, tra cui la posizione della frase nel documento, la lunghezza e la similarità con il titolo del documento.

3.2 Abstractive Summarization

Grazie alla rielaborazione del testo, effettuata tramite tecniche di abstractive summarization, è possibile inserire diversi concetti in una sola frase evitando il fenomeno della ridondanza, verificabile quando vengono estratte intere frasi a priori. La figura 3.1 riassume le tecniche esistenti.

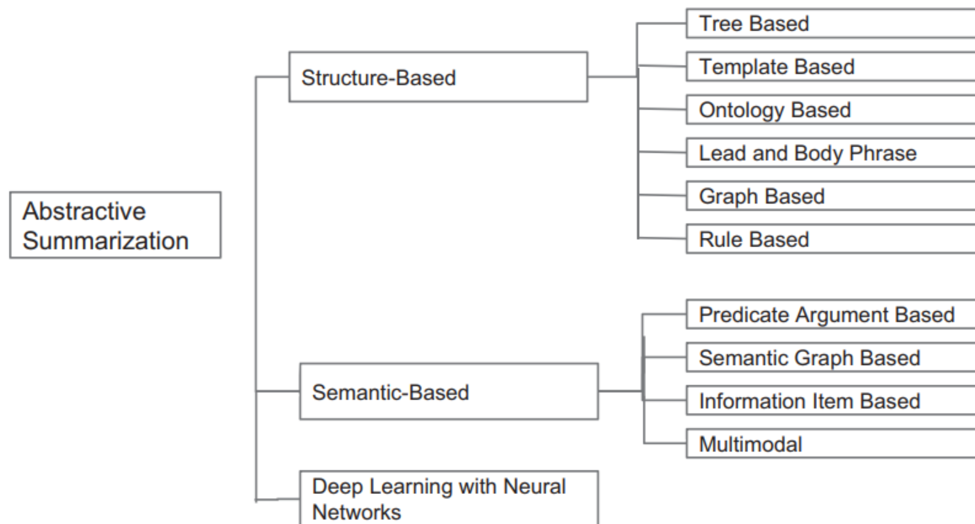


Figura 3.1: Classificazione delle tecniche di abstractive summarization.
Immagine tratta da [13].

Structure-Based Approaches

Si estrapolano le informazioni più importanti e poi, con uno dei seguenti metodi, viene generato il riassunto considerando la struttura della fonte.

- **Tree-based methods**

  Le frasi più importanti vengono disposte in una struttura ad albero in base alla similarità: percorrendolo è possibile generare un predicato che riassume i concetti più vicini tra loro.

- **Template-based methods**

  Vengono estratti dei frammenti del testo e utilizzati all'interno dei template, che permettono di generare dei riassunti con una struttura predefinita. Questo approccio è particolarmente efficace quando il testo ha una certa forma ma, allo stesso tempo, è decisamente oneroso implementare le regole che ne permettono il funzionamento.

- **Ontology-based methods**

  L'ontologia è una rappresentazione formale del dominio di interesse, che ne descrive le entità e le loro relazioni. In questo caso, è definita come un insieme di vocabolari, relazioni e definizioni dei termini: viene sfruttata per estrarre i concetti dal testo e riformularli nel riassunto generato (e.g.,

WordNet [14]). É quindi una collezione di informazioni che permette il riuso della conoscenza.

- **Lead and body phrase methods**

  L'introduzione viene riscritta effettuando inserimenti e sostituzioni di frasi che seguono la struttura sintattica degli incipit delle frasi appartenenti alla testa e al corpo dell'intero testo. Si ottiene così un riassunto, in quanto contiene diverse informazioni estrapolate dal testo.

- **Graph-based methods**

  Il testo viene rappresentato attraverso una struttura a grafo, che aiuta a individuare la similarità tra le frasi. Frasi che si assomigliano contribuiscono a costruire la stessa parte di riassunto.

- **Rule-based methods**

  Il testo viene categorizzato in base ai concetti presenti in esso. Successivamente, vengono posti dei quesiti, ossia delle domande predefinite, per formulare un riassunto finale. Questi quesiti sono pre-definiti manualmente, come per i template-based methods.

Semantic-based approach

Per applicare questa categoria di metodi viene, in principio, costruita una rappresentazione semantica del testo, ossia basata sui concetti e sul significato del testo. Successivamente, questa rappresentazione viene data in input ad un sistema di *Natural Language Generation* per generare il riassunto finale.

- **Information-item-based methods**

  Il riassunto viene generato a seguito di un'analisi delle unità informative del testo, cioè i concetti più elementari che possono dare un contributo alla semantica del riassunto.

- **Predicate-argument based approach**

  Attraverso questo metodo, si estraggono i predicati dal testo, ossia sequenze di testo che contengono soggetto, verbo ed oggetto. Successivamente, vengono definite relazioni di similarità fra queste entità: i predicati semanticamente vicini vengono utilizzati per riformulare lo stesso concetto.

- **Semantic graph-based methods**

  Le relazioni semantiche e sintattiche del testo vengono rappresentate attraverso un grafo. Quelle semantiche vengono estratte sulla base di

un'ontologia, mentre quelle sintattiche sulla base dei legami che vi sono tra soggetti, verbi e oggetti.

- **Multimodal semantic method**

  La rappresentazione semantica del testo viene arricchita da informazioni estratte da diverse fonti (immagini, video, etc.), che si possono trovare, per esempio, nei siti web, che non utilizzano solo il testo per esprimere dei concetti. Il riassunto finale è quindi il risultato dell'elaborazione di diverse fonti che esprimono le stesse idee.

Deep Learning and Neural Network based approach

Le Deep Neural Network sfruttano diversi layer per estrarre informazioni dal testo, dette *features*. Nel tempo si sono sviluppate diverse tipologie di reti:

- **Convolutional Neural Networks (CNN)**: vedi sezione 2.5.2.
- **Recurrent Neural Networks (RNN)**: vedi sezione 2.5.1.
- **Long Short-Term Memories (LSTM)**: vedi sezione 2.5.1.
- **Gated Recurrent Units (GRU)**: vedi sezione 2.5.1
- **Bi-Directional RNN/LSTM/GRU**: considerano due sequenze per generare l'output, una in avanti (*forward direction*) e una indietro.
- **Encoder-Decoder models**: vedi sezione 2.5.3.
- **Transformers**: vedi sezione 2.5.5.

Il paradigma dominante per addestrare modelli a fare Abstractive Summarization, è il **sequence-to-sequence learning**, dove una rete neurale impara a mappare una sequenza di input in una sequenza di output. Recentemente, i Transformers hanno ottenuto lo stato dell'arte e i più noti sono: *BART* [15], *BERT-GPT* [16, 17], *Pegasus* [18], *T5* [19] e *ProphetNet* [20].

Dataset

I principali dataset per addestrare e valutare sistemi di summarization sono:

- *Gigaword*: prodotto dal Stanford University Linguistics Department, consiste in milioni di articoli provenienti da diverse testate giornalistiche.
- *CNN/DailyMail*: consiste in centinaia di migliaia di articoli della CNN e del The Daily Mail.

- *ACL Anthology Reference Corpus*: è una collezione di 10.920 paper accademici provenienti dall'ACL Anthology.
- *DUC2002* e *DUC2004*: prodotti per la Document Understanding Conference al fine di valutare i modelli di summarization. Consistono ciascuno in circa 500 articoli.
- *NYT Annotated Corpus*: consiste in 1.8 milioni di articoli del New York Times pubblicati tra il 1987 e il 2007.
- *Newsroom*: consiste in 1.3 milioni di coppie articolo-riassunti delle maggiori pubblicazioni tra il 1998 e il 2017.
- *XSum (Extreme Summarization)*: consiste in 227k articoli della BBC dal 2010 al 2017.

3.3 Metriche di valutazione

La bontà di un riassunto può essere valutata manualmente (*human evaluation*) o automaticamente. I principali metodi di valutazione automatica vengono descritti di seguito [21].

ROUGE

ROUGE è l'acronimo di *Recall-Oriented Understudy for Gisting Evaluation* ed indica un insieme di metriche di valutazione ampiamente utilizzate nei task di NLP. Lo scopo è quello di comparare un riassunto *machine-produced*, ossia generato da una macchina, e un riassunto target *human-produced*, per misurare la bontà del primo. Le metriche più utilizzate sono *ROUGE-N* e *ROUGE-L*.

$$ROUGE-N = \frac{\sum_{S \in \{Ref\}} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S \in \{Ref\}} \sum_{gram_n \in S} Count(gram_n)}$$

Nel caso di **ROUGE-N** *(ROUGE with N-gram Co-Occurrence Statistics)*, l'obiettivo è calcolare una percentuale del numero di n-gram, che corrispondono (match) tra il riassunto candidato e quello di riferimento.

ROUGE-L *(ROUGE with Longest Common Subsequence)*, invece, viene utilizzato per quantificare la sequenza più lunga in comune tra i due riassunti. In questo caso, la lunghezza dell'n-gram viene decisa automaticamente.

Esistono altre metriche ROUGE, utilizzate raramente:

- **ROUGE-W** (*ROUGE with weighted longest common subsequence*): tra due frasi con lo stesso ROUGE-L, viene scelta quella che ha il numero più alto di parole consecutive corrispondenti a quelle del riassunto target.

- **ROUGE-S** (*ROUGE with Skip-Bigram Co-Occurrence Statistics*): misura il numero di bi-gram, suddivisi da un numero arbitrario di parole, che corrispondono a bi-gram del riassunto reale.

- **ROUGE for Multiple References**: viene applicata quando vi sono n riassunti target. Il risultato finale è la media di n punteggi.

- **ROUGE-WE** (*ROUGE evaluation with Word Embeddings*): misura la similarità tra il riassunto generato e quello predetto nello spazio degli embeddings. Grazie a ciò, non viene considerata solo la similarità letterale, ma anche la semantica.

Information Theory based Metrics

Sono un gruppo di metriche che valutano il potere informativo del testo generato. Tra queste vi sono:

- **Redundancy** [22]: misura quanto le unità semantiche, quindi le informazioni, si ripetono nel riassunto.

  - **Unique n-gram ratio**: misura quanti n-gram univoci ci sono nel documento. Più è basso, più ridondanza vi è nel documento.

$$Uniq_ngram_ratio = \frac{count(uniq_n_gram)}{count(n_gram)}$$

  - **Normalized Inverse of Diversity (NID)**: altra metrica per la ridondanza. *entropy(D)* è una misura dell'eterogeneità che vi è nel documento, quindi del numero di *unigrams* nel documento, che viene normalizzata con il termine *log(|D|)*. Se il fattore di diversità è basso, allora il valore di NID sarà alto, indicando maggior ridondanza.

$$NID = 1 - \frac{entropy(D)}{log(|D|)}$$

  In tabella 3.2 si può osservare che le sequenze di testo più lunghe sono affette da maggior ridondanza, in quanto è più probabile che una certa parola o un certo frammento di testo si possano ripetere.

Datasets	# Doc.	# words/doc.	# words/sent.	NID
Xsum	203k	429	22.8	0.188
CNNDM	270k	823	19.9	0.205
Pubmed	115k	3142	35.1	0.255
arXiv	201k	6081	29.2	0.267

Figura 3.2: La lunghezza del documento influenza il fattore di ridondanza.
Immagine tratta da [22].

- **Relevance**: misura quanto il riassunto generato contiene informazioni relative al testo originale.

- **Informativeness**: misura l'originalità delle informazioni inserite nel riassunto, in modo da non ripetere concetti che il lettore conosce già.

Si può ottenere un buon riassunto massimizzando la seguente funzione:

$$\theta_I(Sum, D, BK) \equiv -\,\mathrm{Red}(Sum) + \alpha\,\mathrm{Rel}(Sum, D) + \beta\,\mathrm{Inf}(Sum, BK)$$

Altre metriche di valutazione

Sono metriche che hanno ricevuto meno attenzione rispetto alle sopracitate. Tra queste vi sono *Perplexity*, *Pyramid*, *Responsiveness*, *SUPERT* e *Preferences-based metric.*

3.4 Multi-document Summarization

La *multi-documenti summarization* (MDS) può essere definita come un particolare task di single-document summarization. In questo caso, l'obbiettivo è generare un riassunto a partire da un insieme di n documenti, che trattano lo stesso argomento, catturando gli aspetti più cruciali ed evitando di restituire informazioni ridondanti. D'altra parte, produrre un riassunto, considerando molteplici documenti con autori diversi e di conseguenza punti di vista differenti, apporta benefici alla comprensibilità e alla precisione del risultato. A differenza della SDS, sono pochi i dataset disponibili. Quelli esistenti trattano notizie (*Multi-news*), articoli di Wikipedia (*Wikisum*), paper scientifici (*SciSumm*) o recensioni (*Oposum*, *Opinosis*, *Yelp*).

3.4.1 Classificazione delle tecniche

É possibile fare una classificazione delle tecniche di MDS basandosi su un survey [21], che fornisce una tassonomia diligente della materia (figura 3.3).

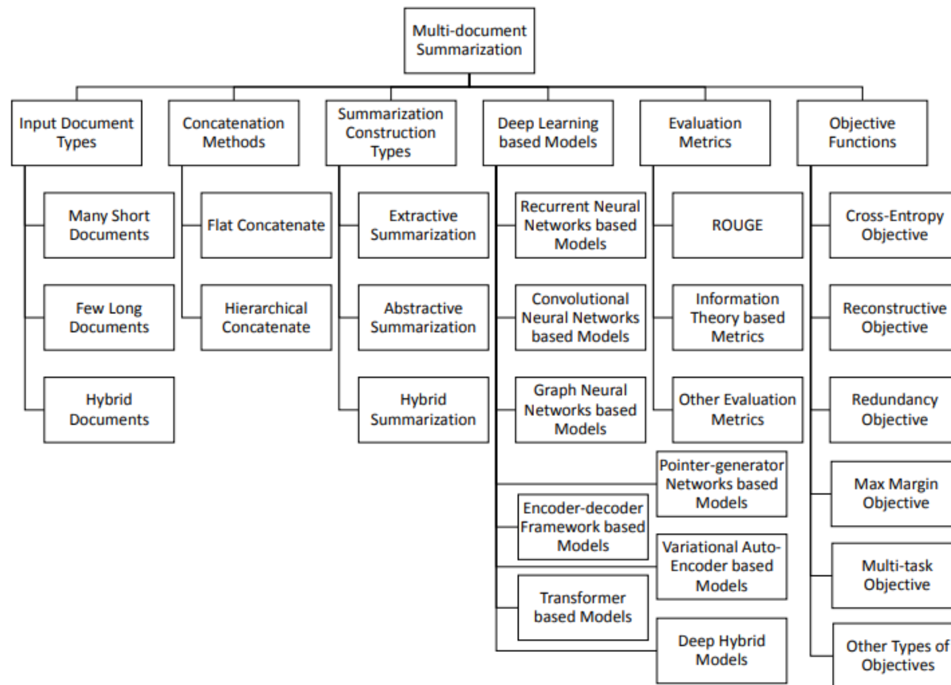


Figura 3.3: Tassonomia delle tecniche di multi-document summarization.
Immagine tratta da [21].

Nello studio [21], sono stati analizzati 30 dei principali lavori pubblicati sull'argomento e sono stati evidenziati diversi aspetti distintivi, descritti nelle prossime sezioni.

Tipo del documento in input

- *Documenti molto corti*: generalmente l'insieme in input è molto grande (e.g., recensioni di prodotti).
- *Documenti lunghi*: generalmente l'insieme è piccolo (e.g., articoli di giornale riguardo una certa notizia).
- *Documenti lunghi* a cui sono *correlati* una serie di documenti più corti (e.g., paper scientifici e le corrispondenti citazioni).

Metodo di concatenazione

- *Flat concatenation*: i documenti vengono concatenati e processati come un'unica sequenza, similmente alla SDS. In questo caso, il modello deve avere la capacità di processare lunghe sequenze.

- *Hierarchical concatenation*: viene prodotto un grafo per modellare le relazioni semantiche tra i documenti, permettendo al modello di selezionare informazioni non ridondanti. Esistono le versioni document-level e word/sentence-level del metodo.

Approccio per la creazione del riassunto

- *Extractive summarization*: vedi sezione 3.1.

- *Abstractive summarization*: vedi sezione 3.2.

- *Hybrid summarization*: si combinano i due approcci precedenti per sfruttarne tutti i vantaggi. Un primo step permette di estrapolare le informazioni, riducendo la lunghezza dell'input del secondo step, che si occuperà di generare il riassunto finale. Esistono due tipi di combinazioni (figura 3.4): Extractive-Abstractive Model e Abstractive-Abstractive Model.

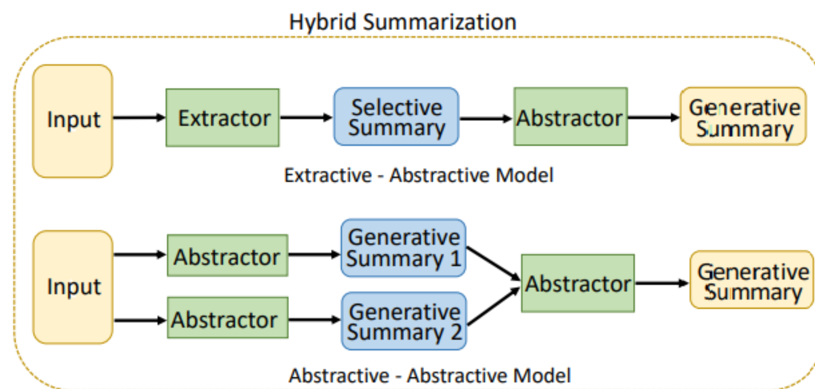
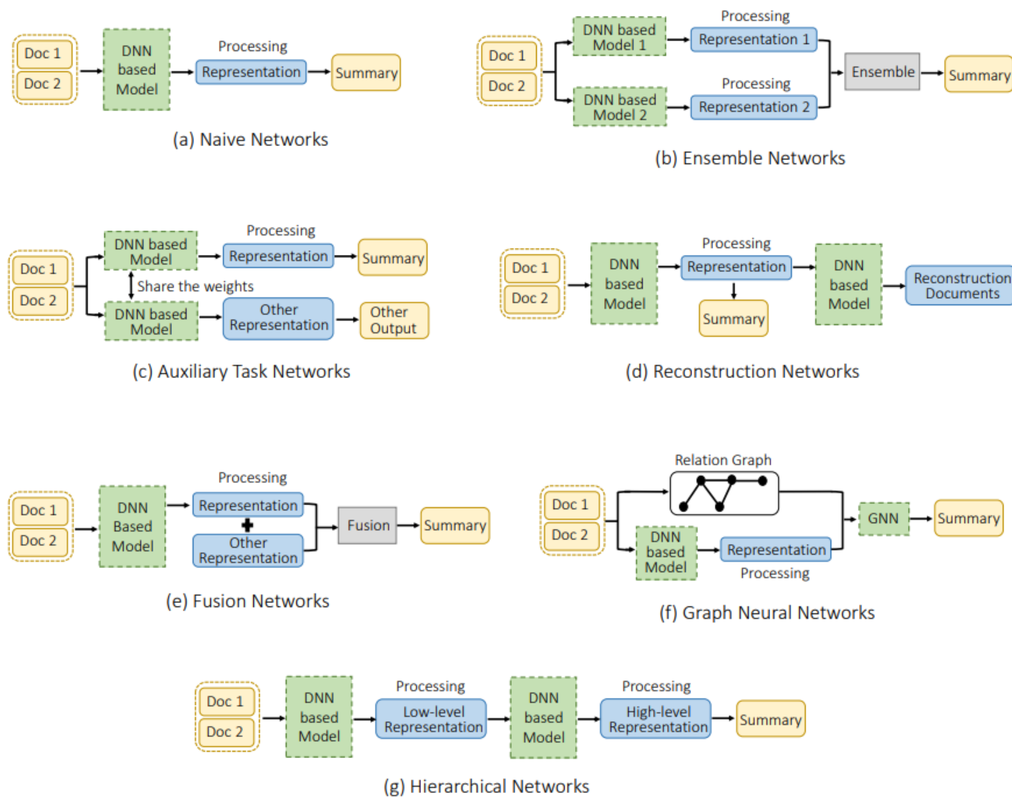


Figura 3.4: Architettura dei sistemi ibridi per la summarization.
Immagine tratta da [21].

Architettura

L'architettura può dipendere dal modo in cui vengono combinate le rappresentazioni prodotte dalle Deep Neural Network (DNN). Nella figura 3.5 sono descritte le possibili strategie.



(a) Naive Networks

(b) Ensemble Networks

(c) Auxiliary Task Networks

(d) Reconstruction Networks

(e) Fusion Networks

(f) Graph Neural Networks

(g) Hierarchical Networks

Figura 3.5: Strategie di design per sistemi MDS.
Immagine tratta da [21].

Modello di Deep Learning

- *Recurrent Neural Network* (RNN): ottima per processare lunghe sequenze di dati, ma non parallelizzabile.

- *Convolutional Neural Network* (CNN): applicando dei filtri di varia lunghezza sulla sequenza in input, è possibile ottenere una rappresentazione della semantica del testo. Inoltre, la rete convoluzionale è parallelizzabile, quindi la fase di training è più efficiente.

- *Graph Neural Networks Based Models*: queste reti modellano, mediante dei grafi, le relazioni semantiche e sintattiche tra parole, frasi o documenti. Allo stesso tempo, i documenti in input vengono affidati a un modello DNN per generare gli embedding.

- *Pointer-generator Networks Based Models*: sono progettati per evitare la ridondanza delle informazioni. Viene introdotto il metodo *Maximal Marginal Relevance* (MMR), che ha l'obiettivo di selezionare un insieme

di frasi dal documento in input, considerando degli indici di importanza e ridondanza: verranno scelte le frasi che hanno una minor sovrapposizione con il riassunto generato fino a quel momento.

- *Encoder-decoder Based Models*: questo modello è composto da due elementi, ossia l'encoder e il decoder. Il primo si occupa di produrre delle rappresentazioni compresse dei documenti in input. Il secondo genera il riassunto a partire dalle rappresentazioni del primo.
- *Variational Auto-Encoder Based Models* (VAE): il modello auto-encoder ha l'obiettivo di imparare una trasformazione dei dati da uno spazio ad alta dimensionalità ad uno spazio a bassa dimensionalità, ottenendo un vettore latente. Nelle reti VAE, lo step successivo permette di riportare il vettore latente in uno spazio a distribuzione normale per permettere la generazione del riassunto (figura 3.6).

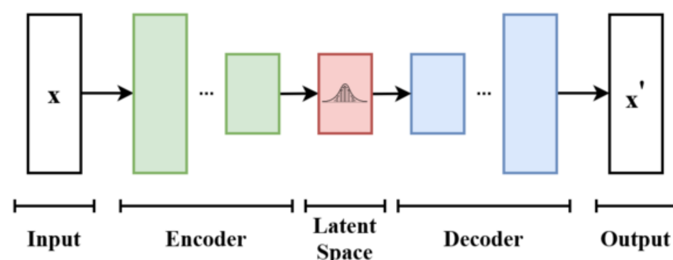


Figura 3.6: Schema del funzionamento di un Variational Auto-Encoder.
Immagine tratta da wikipedia.org.

- *Transformer Based Models*: grazie al meccanismo di attenzione, i Transformers sono parallelizzabili e possono processare lunghi testi.
- *Modelli ibridi*: si integrano più modelli per ottenere prestazioni migliori.

Funzione obiettivo

La funzione di perdita (loss function) viene utilizzata nei sistemi di summarization per calcolare la similarità tra il riassunto prodotto dal modello e il riassunto target, con lo scopo di ottimizzare il modello e aggiornare i parametri attraverso la backpropagation. Esistono diverse funzioni obiettivo per MDS:

- *Cross-Entropy Objective*

$$L_{CE} = -\sum_{i=1} \mathbf{y_i} \log(\mathbf{\hat{y}_i})$$

L'obiettivo è minimizzare la distanza tra il vettore $\mathbf{y_i}$ del riassunto target e il vettore $\mathbf{\hat{y}_i}$ del riassunto generato.

- *Reconstructive Objective*

$$L_{Rec} = D[\mathbf{x_i}, \phi'(\phi(\mathbf{x_i}; \theta); \theta')]$$

L'obiettivo è minimizzare la distanza (calcolata da una funzione D che generalmente è $\mathbf{L_0}$, $\mathbf{L_1}$, $\mathbf{L_2}$) tra il vettore di input $\mathbf{x_i}$ e la ricostruzione $\phi'(\phi(\mathbf{x_i}; \theta); \theta')$ generata da un modello encoder ϕ + decoder ϕ'.

- *Redundancy Objective*

$$L_{Red} = Sim(\mathbf{x_i}, \mathbf{x_j})$$

L'obiettivo è minimizzare la sovrapposizione tra sezioni semantiche del riassunto generato, in modo da spronare il modello a massimizzare la copertura delle informazioni.

- *Max Margin Objective*

$$L_{Margin} = \max(0, f(\mathbf{x_i}; \theta) - f(\mathbf{x_j}; \theta) + \gamma)$$

L'obiettivo è massimizzare la capacità del modello di produrre rappresentazioni distintive. Le funzioni $f(\mathbf{x_i}; \theta)$ e $f(\mathbf{x_j}; \theta)$ sui vettori di input $\mathbf{x_i}$ e $\mathbf{x_j}$ saranno forzate ad essere separate da un margine predefinito γ.

- *Multi-task Objective*

$$L_{Mul} = L_{Summ} + L_{Other}$$

Le funzioni obiettivo per la MDS potrebbero non essere abbastanza per addestrare i modelli a produrre delle rappresentazioni ideali. Possono quindi essere aggiunte funzioni di perdita di task ausiliari (e.g., multi-task classification).

Metriche di valutazione

Le metriche di valutazione descritte nella sezione 3.3 sono valide per misurare la bontà dei sistemi di Multi-document Summarization.

3.4.2 Sviluppi futuri

Il campo della multi-document summarization ha numerosi punti irrisolti, che è possibile approfondire [21]. Per citarne alcuni:

- Catturare le relazioni cross-document, attraverso modelli deep learning combinati con *graphical models*.
- Applicare il *Reinforcement Learning*.
- Creare la disponibilità di modelli pre-addestrati: ad oggi, i sistemi più promettenti sono quelli capaci di processare lunghe sequenze, come *Longformer* [23], *Reformer* [24] e *Big Bird* [25].
- Migliorare l'interpretabilità dei modelli.
- Sviluppare sistemi *multi-modality*, cioè algoritmi che vengono applicati a testo + immagine, testo + video o testo + audio, in modo da catturare informazioni da entrambe le risorse e generare un unico riassunto.
- Aumentare la disponibilità di dataset per MDS.

Capitolo 4

Progetto

4.1 Introduzione e obiettivi

Nel capitolo precedente sono state analizzate le tecniche esistenti di Automatic Text Summarization. Le deep neural network, in particolare i Transformer, rappresentano lo stato dell'arte in ambito NLP, ma sono ancora poche le soluzioni che permettono di processare **lunghe sequenze di testo**: tra queste sono già state citate Reformer, Longformer e BigBird.

Tuttavia, di recente, sono stati proposti metodi per arricchire i modelli Transformer di una **memoria** addestrabile: *Memory Transformer* [26], *Memformer* [27] e *Sliding Selector Network with Dynamic Memory* [28]. Per esempio, nello studio "Memory Transformer" vengono proposte le seguenti estensioni al modello Transformer:

1. Aggiungere memory-tokens per memorizzare una rappresentazione globale del contesto.

2. Creare un filtro per la memoria globale.

3. Controllare l'aggiornamento della memoria con un layer dedicato.

L'obiettivo di questo progetto è l'applicazione di una rete con memoria per fare *abstractive summarization* di lunghe sequenze di testo.

Dato il successo del modello ***RAG*** [29] per i task di *Question Answering*, si è deciso di sperimentare una nuova soluzione basata su esso, al fine di valutare le performance e la capacità di generalizzazione su un nuovo task. RAG (*Retrieval-Augmented Generation*) è un'architettura, introdotta nell'Aprile 2021 dai team di ricerca Facebook AI Research, University College London e New York University, che combina una memoria parametrica e una memoria non-parametrica per i task di *language generation* (vedi sezione 4.1.1).

4.1.1 Differenza fra modelli parametrici e non-parametrici

I *neural language models* (modelli di deep learning per il linguaggio naturale) hanno dimostrato che, attraverso l'aggiornamento dei parametri durante la fase di addestramento, è possibile ottenere dai dati una profonda base di conoscenza e conseguire risultati allo stato dell'arte quando vengono ri-addestrati (fine-tuning) per le applicazioni *downstream* di NLP. Questi modelli vengono definiti con una memoria parametrica (*parametric memory*). Un *parametric model* è un modello che impara a mappare i dati in input in dati di output attraverso una funzione che ha una forma predefinita e quindi un numero di parametri determinato a priori: quest'ultimi definiscono la memoria parametrica. Il vantaggio di questa tipologia di modelli è l'interpretabilità ma, d'altro canto, sono limitati da una regola specifica.

In contrapposizione, esistono i *non-parametric models*, cioè modelli che non vengono addestrati, ma che possono espandere la memoria dei modelli parametrici se combinati con questi. Esistono altri lavori che sfruttano la combinazione di queste due tipologie di memoria: *REALM* [30] e *ORQA* [31].

4.1.2 Formulazione della soluzione

La soluzione proposta in questa tesi consiste nel modificare l'architettura RAG al fine di renderla adatta al task di abstractive summarization. In particolare, si vuole sfruttare e testare la memoria non parametrica del modello, con lo scopo di arricchire la rappresentazione del testo di input che deve essere riassunto. A tal fine, date le grandi dimensioni dei documenti da riassumere, gli input del modello saranno le sottoparti dell'intera sequenza di testo, chiamati *chunk*, in modo da poter processare interi documenti, senza troncare l'input a causa dei vincoli computazionali dell'hardware a disposizione. Allo stesso tempo, si vuole avere "in memoria" il resto del documento, in modo da arricchire la rappresentazione del chunk corrente. Infine, le sequenze generate in output dal modello saranno concatenate in base al documento di origine, per ricostruire il riassunto finale.

4.2 Panoramica delle architetture

4.2.1 RAG

Il modello RAG (figura 4.1) si compone di due parti: il *Dense Passage Retriever* (DPR) [32] e il *Generator*.

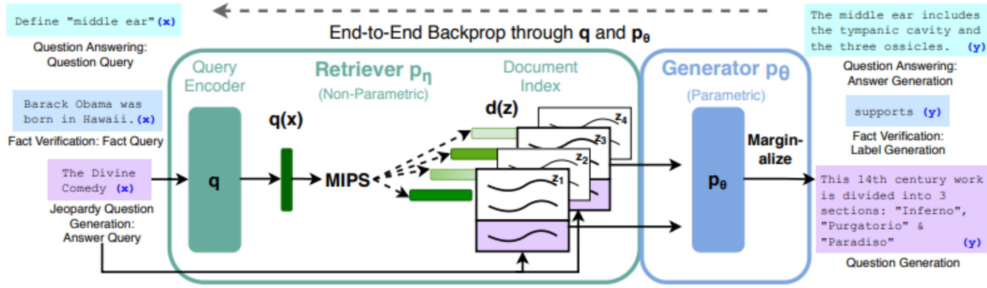


Figura 4.1: Architettura del modello RAG.
Immagine tratta da [29].

Il primo componente (DPR) prende in input una query testuale, che viene codificata dal *Query Encoder*, basato su BERT-base [16], producendo una rappresentazione *q(x)* (con *dim*=768). Contemporaneamente, il DPR produce, attraverso il *Document Encoder* (anch'esso basato su BERT-base), una rappresentazione *d(x)* (con *dim*=768) dei passaggi [1] (blocchi di 100 parole) dei documenti, nei quali il modello deve cercare informazioni per generare una propria risposta: in questo caso in un dataset di articoli di Wikipedia.

L'obiettivo è creare uno spazio vettoriale in cui le coppie *query-passage* rilevanti siano più vicine. A tal fine, si addestrano i modelli encoder a produrre una migliore rappresentazione (*embedding*) delle query e dei passaggi. Dati una query, un *positive passage* (rilevante) e *n negative passages* (non rilevanti), si può ottimizzare la seguente funzione di errore (*negative log-likelihood*):

$$
\begin{aligned}
& L(q_i, p_i^+, p_{i,1}^-, \cdots, p_{i,n}^-) \\
= \; & -\log \frac{e^{\mathrm{sim}(q_i, p_i^+)}}{e^{\mathrm{sim}(q_i, p_i^+)} + \sum_{j=1}^{n} e^{\mathrm{sim}(q_i, p_{i,j}^-)}}.
\end{aligned}
$$

Il DPR e il generatore sono stati addestrati *end-to-end*, quindi non vi è stato nessun training supervisionato per quanto riguarda i documenti che il retriever dovrebbe recuperare. Infatti, esso costituisce la memoria non-parametrica del modello. I documenti sono stati indicizzati utilizzando FAISS [33], ossia una libreria molto efficiente per la *similarity search* e il *clustering* di *dense vectors*.

Una volta ottenute le rappresentazioni della query e dei documenti, è possibile calcolare una lista di k documenti z che ottengono il valore più alto di $p_\eta(z|x)$ (*Maximum Inner Product Search* - MIPS), ossia quei documenti che hanno più probabilità di rispondere correttamente alla query.

$$p_\eta(z|x) \propto \exp\left(d(z)^\top q(x)\right)$$

[1] D'ora in poi, i termini "passaggio" e "documento" sono utilizzati in modo intercambiabile.

I parametri η aggiornano solo il *Query Encoder*, in quanto i ricercatori hanno scoperto che non si hanno significativi miglioramenti nelle performance nell'addestrare il *Document Encoder*.

Il secondo componente, il generatore, è stato modellato con BART-large [15], ma poteva essere un qualsiasi modello encoder-decoder. Esso riceve in input k risultati z del retriever, ognuno concatenato con la query di input, e produce k sequenze in output. L'obiettivo è minimizzare, per l'insieme degli output, la seguente loss (*negative marginal log-likelihood*): $\sum_j -\log p(y_j|x_j)$.

Vi sono due versioni del modello RAG:

- **RAG-Sequence Model**

  Ciascun documento viene utilizzato per generare una sequenza di token in output. Per ognuna di queste viene calcolata una probabilità.

$$p_{\text{RAG-Sequence}}(y|x) \approx \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x)\, p_\theta(y|x,z) =$$

$$\sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) \prod_i^N p_\theta(y_i|x,z,y_{1:i-1})$$

- **RAG-Token Model**

  Il generatore può estrarre informazioni da più documenti per produrre una risposta, quindi viene definita una probabilità per il prossimo token a partire da ognuno dei documenti.

$$p_{\text{RAG-Token}}(y|x) \approx \prod_i^N \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x)\, p_\theta(y_i|x,z,y_{1:i-1})$$

I parametri θ vengono aggiornati per addestrare il generatore e rappresentano la memoria parametrica del modello.

4.2.2 BERT

BERT (*Bidirectional Encoder Representations from Transformers*) è un modello *self-supervised* per la rappresentazione del linguaggio, proposto dal team Google AI Language nel 2019 [16]. Questo sistema implementa un meccanismo di codifica "bidirezionale", che permette di processare simultaneamente i termini di una sequenza e di avere una visione generale della frase in input. Al fine di pre-addestrare BERT, sono stati utilizzati due task semi-supervisionati:

- **Masked Language Modeling (MLM)**

  Questa tecnica prevede che il 15% delle parole della sequenza venga mascherato randomicamente e sostituito con un token `[MASK]`, per questo viene definito *Masked Language Modeling*. L'obiettivo è addestrare BERT a prevedere i token mancanti, basandosi sul significato delle parole non mascherate, quindi sull'intero contesto della frase.

- **Next Sentence Prediction (NSP)**

  Al fine di addestrare BERT a individuare le relazioni tra le frasi, viene applicato un task di *Next Sentence Prediction*. Per ogni istanza del training set, viene estratta una coppia di frasi consecutive. Solo nel 50% delle coppie la seconda stringa rappresenta la reale frase successiva del testo iniziale, nella restante parte si tratta di una frase scelta casualmente all'interno del documento. Il modello viene così addestrato a capire se la seconda frase di ogni coppia corrisponda alla reale proposizione immediatamente seguente alla prima.

BERT è un modello **encoder**, in quanto produce una rappresentazione del testo. All'input del modello vengono aggiunti i token `[CLS]` e `[SEP]` all'inizio e alla fine (figura 4.2). Poi, l'intera sequenza di token viene convertita in indici. L'output del modello sono n vettori (con dimensione d=768), che definiscono una rappresentazione in uno spazio di dimensione d dei token in input.

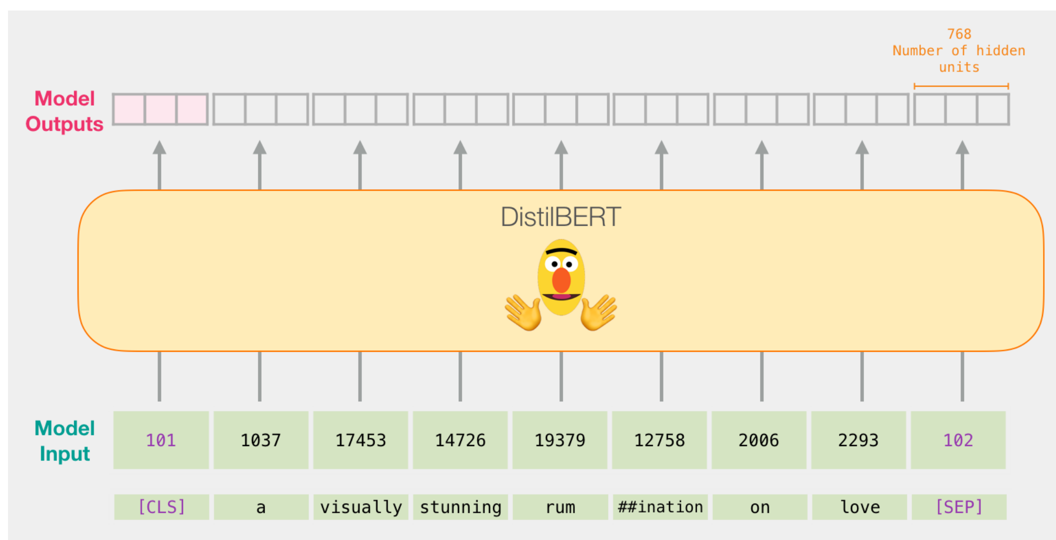


Figura 4.2: Input e output dell'architettura BERT.
Immagine tratta da jalammar.github.io.

Aggiungendo un nuovo layer sopra al modello BERT, è possibile addestrarlo su diversi task specifici di downstream, come la classificazione.

4.2.3 BART

BART è un modello auto-supervisionato, proposto nel 2019 dal team di ricerca Facebook AI [15]. Anche in questo caso, l'obiettivo è ricostruire una sequenza di testo che è stata in parte mascherata (Masked Language Modeling).

A differenza di BERT, il modello BART è un'architettura *seq2seq*, in quanto è composto da un *bidirectional encoder*, che si occupa di corrompere il testo, e un *autoregressive decoder*, con il compito di rigenerare la sequenza (figura 4.3).

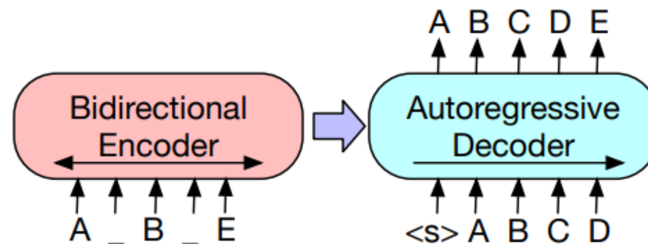


Figura 4.3: Funzionamento dell'architettura BART.
Immagine tratta da [15].

L'encoder applica diverse strategie di mascheramento (figura 4.4), tra cui modificarne la lunghezza; un token `[MASK]` può rimpiazzare una o più parole.

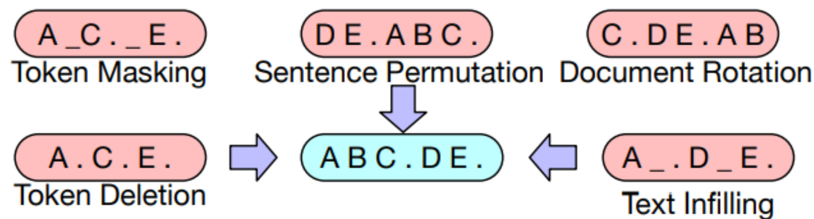


Figura 4.4: Strategie di mascheramento dell'input.
Immagine tratta da [15].

BART viene quindi addestrato a ottimizzare una *reconstruction loss*, ossia una misura dell'accuratezza delle previsioni. É particolarmente indicato per task downstream, come *Sequence Generation* e *Machine Translation*.

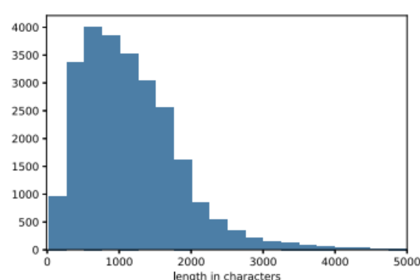
4.3 Analisi del dataset

Per gli esperimenti, è stato utilizzato il dataset *BillSum* [34], che contiene leggi del Congresso degli Stati Uniti e dello stato della California.

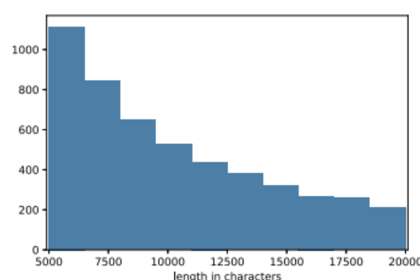
Il primo gruppo di leggi (*US Congressional bills*) è composto da 22.218 testi legali e rispettivi riassunti target. Questo corpora è in seguito suddiviso in training set (18.949 istanze) e test set (3.269 istanze).

Il secondo gruppo (*California state bills*) è un ulteriore test set composto da 1.237 testi di legge e rispettivi riassunti.

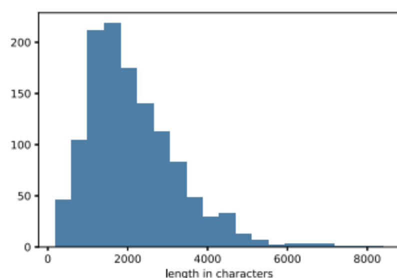
Le leggi esistenti in questi stati sono molte di più, ma la scelta delle istanze, da parte dei creatori del dataset, è ricaduta sui testi lunghi tra i 5.000-20.000 caratteri, quindi quelli di media lunghezza (figura 4.5). I relativi riassunti sono, per il 90% del totale, lunghi al massimo 2.000 caratteri.



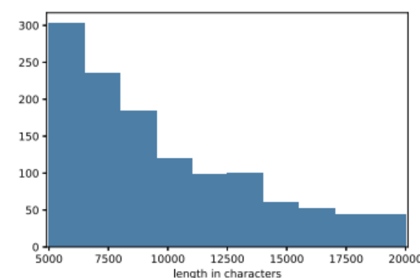
(a) US Bill Summaries



(b) US Bill Text



(c) CA Bill Summaries



(d) CA Bill Text

		mean	min	25th	50th	75th	max
Words	US	1382	245	923	1253	1758	8785
	CA	1684	561	1123	1498	2113	3795
Sentences	US	46	3	31	42	58	372
	CA	47	12	31	42	59	137

Figura 4.5: Distribuzione delle lunghezze dei testi nel dataset *BillSum* rispetto al numero di caratteri, parole e frasi.

Immagine tratta da [34].

Generalmente, ogni legge è introdotta da un breve titolo e continua con

diverse sezioni, ognuna delle quali composta da una o più frasi (figura 4.6).

Summary: This bill authorizes the Department of Education to award competitive grants to nonprofit organizations for the development and implementation of teacher-led projects to improve outcomes in elementary and secondary schools. Grantee organizations shall use grant funds to make competitive subgrants to teachers and school leaders in partnership with the organization or a local educational agency.

SECTION 1. SHORT TITLE.
This Act may be cited as the "Teach to Lead Act of 2016".
SEC. 2. FINDINGS.
Congress finds as follows:
(1) Teachers, because of their position in the classroom, often see important opportunities to improve student learning most directly and thus have a unique perspective from which to create practical solutions to help students succeed.
(2) According to a Scholastic and Bill & Melinda Gates Foundation poll, 69 percent of teachers feel that their voices are heard in their school, but only one-third feel heard in their district, five percent in their State, and two percent at the national level….
SEC. 3. PURPOSE.
The purpose of this Act is to empower teachers to develop and implement projects with the potential to have a wider impact on developing the knowledge, pedagogical skills, and conditions needed to improve teaching and student outcomes, particularly academic growth, by bringing their classroom knowledge and expertise directly to bear on the many challenges confronting our education system.
SEC. 4. GRANT PROGRAM.
(a) In General.—
(1) PROGRAM AUTHORIZED.—From the funds made available under section 7, the Secretary of Education may make grants, on a competitive basis, to one or more nonprofit organizations to award subgrants to eligible entities to develop and implement teacher-led projects to improve teaching and student outcomes in elementary school and secondary school, particularly academic growth.
(2) GRANT PERIOD.—A grant made to a nonprofit organization under paragraph (1) shall be for a period of not more than five years.
(3) USE OF GRANT FUNDS.—A nonprofit organization that receives a grant under paragraph (1)—
(A) shall reserve not less than 90 percent of the grant to award subgrants, on a competitive basis, to eligible entities under subsection (c); and
(B) may use not more than 10 percent of the grant for administrative purposes.
(b) Applications.—A nonprofit organization that desires a grant under this section shall submit an application to the Secretary at such time and in such manner, and containing such information as the Secretary may require. The application shall—
(1) demonstrate the entity's ability to—
(A) operate a national program, a multi-State program, or a program that reaches not less than 100,000 students;
(B) manage the administrative and fiscal aspects of the subgrant program described in this section; …
(c) Subgrants.—
(1) SUBGRANT PRIORITY.—A nonprofit organization receiving a grant under this section shall use such grant to award subgrants to eligible entities under this subsection, and in awarding such subgrants the nonprofit organization shall give priority to eligible entities that will use the subgrants to carry out projects that—
(A) are designed to improve teaching and learning outcomes for all students in high-need schools or that target the educational needs of low-income or minority students;
…
(2) SUBGRANT APPLICATIONS.—An eligible entity that desires a subgrant under this section shall submit an application to the applicable nonprofit organization awarded a grant under this section at such time and in such manner, and containing such information as the nonprofit organization may reasonably require. Each application shall, at a minimum, describe—
(A) the project proposed, including timelines, resources needed, and any measurable objectives to be used in determining how the project will improve teaching and student outcomes, particularly academic growth…
(3) USE OF SUBGRANT FUNDS.—
(A) USE OF SUBGRANT FUNDS.—An eligible entity shall use the subgrant received under this section to develop and implement an innovative project designed and led by teachers, teams of teachers, or teachers and school leaders to improve teaching and learning at the elementary school and secondary school level, such as—
(i) increasing student engagement through personalized learning, including technology-enabled instruction;
(ii) strengthening support for educators, including support for implementation of challenging, academic standards to prepare students to be ready for college and careers….
(B) ADMINISTRATIVE EXPENSES.—A partner local educational agency or nonprofit organization that serves as the fiscal agent for an eligible entity, may use not more than two percent of the subgrant for direct administrative expenses incurred in carrying out its responsibilities under the subgrant.
SEC. 5. PERFORMANCE MEASUREMENT.
The Secretary shall establish goals and performance indicators to measure and assess the impact of the activities carried out under this Act.
SEC. 6. DEFINITIONS.
In this Act:
(1) ELIGIBLE ENTITY.—The term "eligible entity" means an individual teacher, a team of teachers, or teachers and school leaders, in partnership with a local educational agency or a nonprofit organization that serves as the fiscal agent with respect to funds awarded under this Act.
(2) ESEA TERMS.—The terms "elementary school", "secondary school", "local educational agency", and "Secretary" have the meanings given the terms in section 8101 of the Elementary and Secondary Education Act of 1965 (20 U.S.C. 7801).

Figura 4.6: Esempio di istanza del dataset *BillSum*.
Immagine tratta da [34].

4.4 Esperimenti

Al fine di valutare le performance dell'architettura RAG nel task di *abstractive summarization*, sono state sperimentate le seguenti soluzioni:

1. **Chunks come contesto e Se3-BART come generatore**

   Il retriever ricerca, per ogni chunk in input, le corrispondenze in un dataset composto dagli altri *n-1* chunk dello stesso documento.

   Per consentire la concatenazione del chunk in input con ognuno dei chunk restituiti dal DPR, è stata impostato `max_combined_length` a 1024, in quanto l'input del *context encoder* ha lunghezza 512 token, come il chunk in input che deve essere riassunto.

   Il generatore è un modello BART pre-addestrato a fare riassunti sul dataset *BillSum*, ottenuto nell'ultimo lavoro di ricerca del dott. Luca Ragazzi e prof. Gianluca Moro.

2. **Frasi come contesto e Se3-BART come generatore**

   In questa soluzione, il retriever ricerca, per ogni chunk in input, le corrispondenze in un dataset composto dal resto del documento suddiviso in frasi. Per estrarre le frasi, è stato utilizzato il *sentence tokenizer* della libreria *PYSBD (Python Sentence Boundary Disambiguation).*

   In questo caso, la configurazione `max_combined_length` è stata impostata a 768, in quanto l'input ha lunghezza 512 e le frasi del contesto hanno in media lunghezza 46 (figura 4.5).

   Il generatore è lo stesso modello dell'esperimento precedente.

3. **Chunks random come contesto e Se3-BART come generatore**

   In questa soluzione, il retriever ricerca, per ogni chunk in input, le corrispondenze in un dataset composto da chunks aventi un insieme di frasi scelte randomicamente tra il resto del documento. Più precisamente, per ogni input, è stata effettuata la suddivisione del resto del documento nel seguente numero di chunk:

   $$\frac{\text{Numero totale di frasi del documento} - \text{Numero di frasi dell'input}}{2}$$

   Il numero di chunk nel retriever è la metà del numero di frasi rimanenti e, per ogni chunk, vengono scelte 5 frasi casualmente. É quindi molto probabile che una frase si ripresenti in diversi chunk. É stata scelta questa soluzione poiché un riassunto di un chunk potrebbe essere influenzato da

diverse frasi sparse all'interno del documento. Creare quindi molti chunk con frasi prese randomicamente dall'intero testo, comporta la formazione di possibili chunks utili al riassunto del chunk i-esimo.

Le frasi possono avere lunghezze diverse, perciò `max_combined_length` è stato impostato a 1024.

Il generatore è lo stesso modello degli esperimenti precedenti.

4. **Soluzione con BART-base come generatore**

   Si ripeteranno gli esperimenti con la soluzione che risulta essere migliore, ma utilizzando come generatore BART-base non addestrato.

Per ognuna di queste soluzioni sono stati effettuati i seguenti esperimenti:

- **RAG (10) + RAG**

  Il modello RAG viene addestrato *end-to-end* su 10 documenti del training set. La valutazione è effettuata sull'intero test set e l'intera architettura RAG (DPR + generatore) viene impiegata per generare le predizioni.

- **RAG (10) + BART**

  Il modello RAG viene addestrato *end-to-end* su 10 documenti del training set. La valutazione è effettuata sull'intero test set e il solo componente generatore (BART) viene impiegato per generare le predizioni.

- **RAG (100) + RAG**

  Il modello RAG viene addestrato *end-to-end* su 100 documenti del training set. La valutazione è effettuata sull'intero test set e l'intera architettura RAG (DPR + generatore) viene impiegata per generare le predizioni.

- **RAG (100) + BART**

  Il modello RAG viene addestrato *end-to-end* su 100 documenti del training set. La valutazione è effettuata sull'intero test set e il solo componente generatore (BART) viene impiegato per generare le predizioni.

Tutti i modelli, per ogni esperimento, sono stati addestrati per 1 epoca.

4.4.1 Discussione dei risultati

Train (*n_docs*) + Eval	**ROUGE** R1 / R2 / RL	**%ngram** uni / bi / tri	**NID**
References			
Source	-	22.50/62.21/82.90	28.09
Target	-	51.20/82.37/91.53	23.78
Baselines w/ Se3-BART			
Se3-BART (*10*)	44.37/21.17/27.57	56.66/88.25/96.39	22.87
Se3-BART (*100*)	47.85/26.67/33.36	59.91/89.52/96.03	21.74
Se3-BART (*full*)	57.66/38.20/44.11	56.43/89.10/96.27	21.58
Chunks come contesto e Se3-BART come generatore			
RAG (*10*) + RAG	50.59/30.73/36.45	46.04/68.09/74.07	28.04
RAG (*10*) + BART	57.00/37.09/42.54	53.06/86.01/93.55	21.84
RAG (*100*) + RAG	49.60/29.93/35.76	46.77/68.18/73.92	28.11
RAG (*100*) + BART	56.74/36.78/42.27	53.60/86.32/93.70	**21.77**
Frasi come contesto e Se3-BART come generatore			
RAG (*10*) + RAG	52.13/32.35/38.40	55.80/82.01/87.65	23.40
RAG (*10*) + BART	57.00/37.09/42.53	53.07/86.03/93.56	21.83
RAG (*100*) + RAG	50.84/31.50/37.64	56.92/82.86/88.30	23.31
RAG (*100*) + BART	**56.78/36.82/42.30**	53.58/86.31/93.70	21.78
Chunks random come contesto e Se3-BART come generatore			
RAG (*10*) + BART	57.01/37.11/42.55	53.07/86.02/93.55	21.84
RAG (*100*) + BART	56.69/36.76/42.26	**53.61/86.32/93.70**	**21.77**
Frasi come contesto e BART-base come generatore			
RAG (*100*) + BART	36.10/18.75/22.92	35.97/74.02/89.38	25.71

Tabella 4.1: Esperimenti con il modello RAG sul dataset BillSum. I risultati migliori sono in grassetto.

La tabella 4.1 riporta i risultati degli esperimenti effettuati. Nella parte alta sono state riportate le statistiche di ridondanza (%n-gram e NID), calcolate rispetto ai documenti e i rispettivi riassunti target nel test set: serviranno come riferimento per un confronto con i valori ottenuti nelle soluzioni proposte.

Successivamente, vengono riportate le metriche delle performance di un modello BART addestrato (fine-tuned) sul dataset BillSum a fare abstractive summarization. I pesi e le configurazioni del modello sono state fornite dal relatore e correlatore di tesi, al fine di caricarle nel generatore di RAG.

Per ogni soluzione proposta, la scelta di generare le predizioni sul test set utilizzando solo il generatore di RAG ha permesso di ottenere risultati migliori

su tutte le metriche. Ciò potrebbe essere dovuto ad una caratteristica intrinseca del modello RAG. Difatti, il retriever effettua la concatenazione, o arricchimento dell'input, ponendo all'inizio il contesto recuperato e successivamente la stringa in input, in quanto, nel task Question Answering, il primo rappresenta la fonte di informazioni sulla quale focalizzarsi per generare la risposta. Questo fatto potrebbe aver condizionato le predizioni generate utilizzando l'architettura RAG: sarebbe necessario un addestramento su più documenti e per più epoche al fine di ottenere l'effetto contrario, ossia permettere al modello di focalizzarsi maggiormente sull'input rispetto al contesto aggiunto. Perciò sono state effettuate delle prove di addestramento sull'intero dataset (per 1 epoca) e si è notato che il modello perde la capacità di generalizzazione nel momento in cui vengono sottoposti tanti esempi di training. Ciò è confermato dai risultati ottenuti addestrando i vari modelli su 100 documenti rispetto a 10 documenti: questo problema può essere migliorato, ma non completamente risolto, attraverso ulteriori studi sul learning rate. Difatti, si è scoperto che, diminuendo progressivamente tale iper-parametro, il modello era capace di generalizzare meglio su un numero più grande di esempi.

Il modello che utilizza le frasi come contesto, invece dei chunk, ha ottenuto risultati migliori per quanto riguarda le soluzioni che utilizzano RAG come generatore nella fase di evaluation. Di conseguenza, si può intuire che arricchendo l'input di un contesto più corto del chunk stesso vengono generati riassunti migliori. D'altra parte, la soluzione che utilizza i chunk come contesto ha riportato i valori di ROUGE e ridondanza peggiori tra gli esperimenti effettuati. Difatti, il retriever concatena all'input un contesto che ottiene il punteggio più alto di similarità (in questo caso similarità coseno), ma ciò può produrre ridondanza se l'input viene addizionato di un intero chunk, il cui riassunto sarà comunque generato in un'altra iterazione e concatenato nella soluzione finale.

Un'ulteriore soluzione al problema della ridondanza, causata dal modello che utilizza i chunk come contesto, è l'utilizzo di chunk formati da frasi randomiche del documento. Ciò forza il DPR a includere nel contesto, con una buona probabilità, delle frasi che non si avvicinano semanticamente al chunk corrente. Per tale soluzione è stato scelto BART come generatore nella fase di evaluation perchè risulta essere nei precedenti esperimenti la configurazione migliore.

Col fine di ridurre la ridondanza, si potrebbe addestrare i parametri del context encoder in modo da produrre rappresentazioni diverse dei documenti (in questo caso chunk o frasi) e fare in modo che i contesti molto simili vengano disposti lontano dal chunk corrente nello spazio vettoriale creato dal DPR. Tuttavia, non è stato possibile effettuare anche l'addestramento di questo modello in quanto, essendo basato su BERT-base, che ha 110 milioni di parametri, non erano sufficienti le risorse hardware a disposizione (1 GPU da 24 GB).

Infine, il modello che sfrutta BART-base come generatore ha riportato

risultati meno buoni rispetto a quelli ottenuti da Se3-BART addestrato sulla stessa porzione di dataset. Ciò indica che, in generale, il modello basato su RAG è migliorabile per il task di abstractive summarization.

4.5 Codice prodotto

4.5.1 Preparazione del dataset

Per consentire l'elaborazione di lunghe sequenze, si è deciso di suddividere ciascun documento in *chunk*, in modo da ridurre la dimensione dell'input del modello tra i 256 e i 512 token. Una volta suddiviso il testo legale, è risultato necessario eseguire una suddivisione del riassunto target in modo da mantenere il binomio *testo-riassunto*, essenziale per l'addestramento supervisionato dei modelli. Questa suddivisione è stata precedentemente effettuata dal dott. Luca Ragazzi per un suo lavoro di ricerca. Il risultato del processo è stato:

- Due dataframe [2], rispettivamente per il training e test set, corrispondenti all'intero elenco di chunk estratti da tutti i documenti presenti nei due dataset. I riassunti target sono stati scelti tra le sottoparti dei riassunti originali in base al grado di similarità con il testo dei chunk.

- Due dataframe, rispettivamente per il training e test set, che definiscono per ogni documento il numero di chunk in cui è stato suddiviso.

Successivamente, è stato necessario definire le seguenti funzioni per mantenere traccia del documento a cui fa riferimento ciascun chunk e per eliminare dal training set i chunk che non avevano un riassunto target.

```
1  def add_doc_index(dataframe_chunked, dataframe_chunked_idx):
2      dataframe_chunked["doc_index"] = 0
3      index = 0
4      for i in range(len(dataframe_chunked_idx)):
5          num_chunks = dataframe_chunked_idx.iloc[i]["idx"]
6          dataframe_chunked.loc[index:index + num_chunks, "doc_index"] = i
7          index += num_chunks
8      return dataframe_chunked
9
10 def remove_chunks_without_target(dataframe_chunked, dataframe_chunked_no_nan,
11         dataframe_chunked_idx, dataframe_chunked_idx_no_nan):
12     index = 0
13     for i in range(len(dataframe_chunked_idx)):
14         num_chunks = dataframe_chunked_idx.iloc[i]["idx"]
15         count_not_nan = dataframe_chunked[index:index + num_chunks]["summary"].count()
16         dataframe_chunked_idx_no_nan.iloc[i]["idx"] = count_not_nan
17         index += num_chunks
18     dataframe_chunked_no_nan = dataframe_chunked_no_nan.dropna()
19     return dataframe_chunked_no_nan, dataframe_chunked_idx_no_nan
```

[2] **Dataframe**: struttura bidimensionale contenente dati etichettati di tipo eterogeneo.

4.5.2 Modifica del RagRetriever

Per gli esperimenti è stata utilizzata un'implementazione dell'architettura RAG resa disponibile dal framework HuggingFace [35].

L'implementazione originale della classe `RagRetriever` prevede che per inizializzare un'istanza del retriever (DPR) sia necessario specificare il dataset di documenti in cui verranno ricercate le corrispondenze con le query, cioè i chunk da riassumere nel nostro caso.

Il nostro obiettivo, però, è di modificare dinamicamente tale dataset in modo che, per ogni input, il retriever calcoli $p_\eta(z|x)$ con x uguale al chunk corrente e z scelto tra le altre parti dello stesso documento. A tal fine, sono state effettuate le modifiche descritte di seguito.

Commentando il codice alle righe 3, 4, 6, 14 e 15 si evita il comportamento di default della funzione `__init__()` del `RagRetriever`.

```
1  def __init__(self, config, question_encoder_tokenizer, generator_tokenizer, index=None,
2          init_retrieval=True):
3      # self._init_retrieval = init_retrieval
4      # requires_backends(self, ["datasets", "faiss"])
5      super().__init__()
6      self.index = index  # or self._build_index(config)
7      self.generator_tokenizer = generator_tokenizer
8      self.question_encoder_tokenizer = question_encoder_tokenizer
9
10     self.n_docs = config.n_docs
11     self.batch_size = config.retrieval_batch_size
12
13     self.config = config
14     # if self._init_retrieval:
15         # self.init_retrieval()
16
17     self.ctx_encoder_tokenizer = None
18     self.return_tokenized_docs = False
```

La creazione del *faiss index* avviene nella funzione `_build_index()` o `init_retrieval()`. Le funzioni `from_pretrained()` e `save_pretrained()` della classe `RagRetriever` consentono di caricare o salvare un'istanza di retriever, insieme al dataset indicizzato. Le funzioni appena citate non sono più utili, in quanto il dataset e l'indice vengono istanziati dinamicamente grazie alla funziona `create_and_set_indexed_dataset()`, che prende in input i testi dei chunks presenti nel documento del chunk corrente. Per costruire il *faiss index*, è necessario che ogni istanza del dataset sia composta da titolo, testo ed embedding. Gli embeddigns vengono creati mediante la funzione `embed()`, dove il *context encoder* codifica i testi dei documenti (con `max_length = 512`, invece di 300). Successivamente, viene settato all'interno delle configurazioni del modello il `dataset_path` e `index_path`.

```python
ctx_encoder_checkpoint = "facebook/dpr-ctx_encoder-multiset-base"
ctx_encoder = DPRContextEncoder.from_pretrained(ctx_encoder_checkpoint).to(device)
ctx_encoder.config.max_length = 512
ctx_tokenizer_checkpoint = "facebook/dpr-ctx_encoder-multiset-base"
ctx_tokenizer = DPRContextEncoderTokenizerFast.from_pretrained(ctx_tokenizer_checkpoint)

new_features = Features(
    {"text":Value("string"), "title":Value("string"), "embeddings":Sequence(Value("float32"))}
)  # optional, save as float32 instead of float64 to save space

def create_and_set_indexed_dataset(chunks):
    chunks_with_titles = []
    for i,chunk in enumerate(chunks):
        title = "title" + str(i)
        row = {"title":title, "text":chunk}
        chunks_with_titles.append(row)

    chunks_df = pd.DataFrame(chunks_with_titles)
    dataset = Dataset.from_pandas(chunks_df) # create the dataset
    dataset = dataset.map(
        partial(embed),
        batched=True,
        batch_size=16,
        features=new_features
    ) # map dataset to new features

    dataset_path = os.path.join("my_knowledge_dataset")
    dataset.save_to_disk(dataset_path)
    # Let's use the Faiss implementation of HNSW for fast approximate nearest neighbor search
    index = faiss.IndexHNSWFlat(768, 128, faiss.METRIC_INNER_PRODUCT)
    # add faiss index to dataset
    dataset = dataset.add_faiss_index("embeddings", custom_index=index)
    index_path = os.path.join("my_knowledge_dataset_hnsw_index.faiss")
    dataset.get_index("embeddings").save(index_path)

    # set dataset and index into model
    model.config.passages_path = dataset_path
    model.config.index_path = index_path

def embed(documents):
    """Compute the DPR embeddings of document"""
    input_ids = ctx_tokenizer(
        documents["title"], documents["text"], truncation=True, padding="longest",
        return_tensors="pt")["input_ids"]
    embeddings = ctx_encoder(input_ids.to(device), return_dict=True).pooler_output
    return {"embeddings": embeddings.detach().cpu().numpy()}
```

Ogni volta che si richiama il metodo `retrieve()` del `RagRetriever`, quindi per ogni input, viene creata un'istanza della classe `CustomHFIndexSum` (riga 4), che prende come argomenti il `dataset_path` e l'`index_path` dalle configurazioni del modello precedentemente aggiornate.

```python
def retrieve(self, question_hidden_states, n_docs):

    self.index = CustomHFIndexSum.load_from_disk(
        vector_size=self.config.retrieval_vector_size,
        dataset_path=self.config.passages_path,
        index_path=self.config.index_path)

    doc_ids, retrieved_doc_embeds = self._main_retrieve(question_hidden_states, n_docs)
    return retrieved_doc_embeds, doc_ids, self.index.get_doc_dicts(doc_ids)
```

La classe `CustomHFIndexSum` è una neo-versione delle classi `CustomHFIndex` e `HFIndexBase` [35]. Il metodo `_check_dataset_format()` verifica che il dataset passato come argomento abbia le colonne *title*, *text* ed *embeddings*, necessarie per la costruzione dell'indice. Il metodo `load_from_disk()` istanzia il dataset non indicizzato caricandolo da disco, mentre il metodo `init_index()` si occupa di aggiungere l'indice, passato come argomento, al dataset. Infine i metodi `get_doc_dicts()` e `get_top_docs()` restituiscono i documenti, che in questo caso sono chunk o frasi, più rilevanti per il chunk in input.

```
1  class CustomHFIndexSum(Index):
2      def __init__(self, vector_size, dataset, index_path,
3              index_initialized=False):
4          self.vector_size = vector_size
5          self.dataset = dataset
6          self.index_path = index_path
7          self.index_initialized = index_initialized
8          self.init_index()
9          self._check_dataset_format(with_index=self.index_initialized)
10         dataset.set_format("numpy", columns=["embeddings"], output_all_columns=True,
11             dtype="float32")
12
13     def _check_dataset_format(self, with_index: bool):
14         if not isinstance(self.dataset, Dataset):
15             raise ValueError(f"Dataset should be a datasets.Dataset object,"
16             "but got {type(self.dataset)}")
17         if len({"title", "text", "embeddings"} - set(self.dataset.column_names)) > 0:
18             raise ValueError(
19                 "Dataset should be a dataset with the following columns: "
20                 "title (str), text (str) and embeddings (arrays of dimension vector_size), "
21                 f"but got columns {self.dataset.column_names}"
22             )
23         if with_index and "embeddings" not in self.dataset.list_indexes():
24             raise ValueError(
25                 "Missing faiss index in the dataset. Make sure you called "
26                 "`dataset.add_faiss_index` to compute it or `dataset.load_faiss_index`"
27                 "to load one from the disk."
28             )
29
30     @classmethod
31     def load_from_disk(cls, vector_size, dataset_path, index_path):
32         if dataset_path is None or index_path is None:
33             raise ValueError(
34                 "Please provide ``dataset_path`` and ``index_path`` after calling"
35                 "``dataset.save_to_disk(dataset_path)`` and"
36                 "``dataset.get_index('embeddings').save(index_path)``."
37             )
38         dataset = load_from_disk(dataset_path)
39         return cls(vector_size=vector_size, dataset=dataset,
40             index_path=index_path)
41
42     def init_index(self):
43         if not self.is_initialized():
44             self.dataset.load_faiss_index("embeddings", file=self.index_path)
45             self.index_initialized = True
46
47     def is_initialized(self):
48         return self.index_initialized
49
50     def get_doc_dicts(self, doc_ids: np.ndarray) -> List[dict]:
51         return [self.dataset[doc_ids[i].tolist()] for i in range(doc_ids.shape[0])]
52
53     def get_top_docs(self, question_hidden_states: np.ndarray, n_docs=5) ->
54             Tuple[np.ndarray, np.ndarray]:
```

```
55         _, ids = self.dataset.search_batch("embeddings", question_hidden_states, n_docs)
56         docs = [self.dataset[[i for i in indices if i >= 0]] for indices in ids]
57         vectors = [doc["embeddings"] for doc in docs]
58         for i in range(len(vectors)):
59             if len(vectors[i]) < n_docs:
60                 vectors[i] = np.vstack([vectors[i], np.zeros((n_docs - len(vectors[i]),
61                     self.vector_size))])
62         return np.array(ids), np.array(vectors)
```

4.5.3 Pre-processamento dei dati

Il metodo `preprocess_function()` si occupa di mappare i chunk e i rispettivi riassunti target negli input che verranno dati al modello RAG in fase di training: *input_ids*, *attention_mask* e *labels*.

Inoltre, alla riga 26 viene creata una lista di "dataset" per il retriever a cui si farà accesso per impostare dinamicamente l'insieme di chunk in cui trovare le corrispondenze con il chunk corrente.

```
1  question_encoder_checkpoint = "facebook/dpr-question_encoder-single-nq-base"
2  question_encoder_tokenizer = AutoTokenizer.from_pretrained(question_encoder_checkpoint)
3  generator_tokenizer = AutoTokenizer.from_pretrained(args.checkpoint_generator)
4  tokenizer = RagTokenizer(question_encoder_tokenizer, generator_tokenizer)
5
6  max_len_source = 512
7  max_len_summary = 128
8
9  def preprocess_function(batch):
10     """Prepares the dataset to be process by the model."""
11     inputs = tokenizer(batch["text"], padding="max_length", max_length=max_len_source,
12         truncation=True)
13     outputs = tokenizer(batch["summary"], padding="max_length", max_length=max_len_summary,
14         truncation=True)
15     batch["input_ids"] = inputs.input_ids
16     batch["attention_mask"] = inputs.attention_mask
17     batch["labels"] = outputs.input_ids
18     batch["labels"] = [[-100 if token == tokenizer.generator.pad_token_id else token
19                         for token in labels] for labels in batch["labels"]]
20     return batch
21
22 train_dataset_model_input = train_dataset.map(
23         preprocess_function,
24         batched=True,
25         batch_size=1,
26         remove_columns=["text", "summary", "__index_level_0__"]
27     )
28
29 train_dataset_model_input = train_dataset_model_input.rename_column("Unnamed: 0", "chunk_id")
30
31 train_dataset_retriever_list = create_dataset_retriever_list(train_dataset_model_input,
32     train_dataset_chunked, args.sentence_mode, args.random_chunks)
33
34 train_dataset_model_input.set_format(
35     type="torch",
36     columns=["input_ids", "attention_mask", "labels", "doc_index", "chunk_id"]
37 )
```

La funzione `create_dataset_retriever_list()` ha un comportamento diverso a seconda della soluzione che si vuole adottare: nel retriever è possibile

utilizzare un dataset di chunk, di frasi oppure di chunk creati con frasi scelte randomicamente (vedi sezione 4.4). Nel caso in cui vengano scelte le ultime due opzioni, sarà necessario suddividere il resto del documento in frasi utilizzando il `sentence_tokenizer`.

```
1  def create_dataset_retriever_list(dataset, dataframe, sentence_mode, random_chunks):
2      if sentence_mode or random_chunks:
3          # set up the tokenizer to split the sentences
4          sentences_tokenizer = pysbd.Segmenter(language="en", clean=True)
5      dataset_retriever_list = []
6      # for each chunk to summarize
7      for elem in dataset:
8          # retrieve the other chunks of the same document
9          chunks_filtered = dataframe[dataframe["doc_index"] == elem["doc_index"]]
10         # if only 1 chunk is retrieved, we do not filter again
11         if len(chunks_filtered) > 1:
12             chunks_filtered = chunks_filtered[chunks_filtered["Unnamed: 0"]!=elem["chunk_id"]]
13         chunks_filtered = chunks_filtered["text"]
14         if random_chunks:
15             chunks_filtered = pd.Series(create_random_chunks(chunks_filtered,
16                 sentences_tokenizer))
17         # for each chunk retrieved, we take its sentences
18         elif sentence_mode:
19             all_sentences = []
20             for chunk in chunks_filtered:
21                 sentences = sentences_tokenizer.segment(chunk)
22                 all_sentences.extend(sentences)
23             chunks_filtered = pd.Series(all_sentences)
24         dataset_retriever_list.append(chunks_filtered)
25     return dataset_retriever_list
26
27
28 def create_random_chunks(chunks, sentences_tokenizer, f=5):
29     all_sentences = []
30     for chunk in chunks:
31         sentences = sentences_tokenizer.segment(chunk)
32         all_sentences.extend(sentences)
33     f = len(all_sentences) if len(all_sentences) < f else f
34     random_chunks = [" ".join(random.sample(all_sentences, f))
35                      for _ in range(len(all_sentences)//2)]
36     return random_chunks
```

4.5.4 Training

Il modello RAG viene istanziato a partire da due modelli pre-addestrati: il *question encoder* e il *generator* che, nei nostri esperimenti, può essere un modello BART già addestrato a fare riassunti (Se3-BART) o un modello non addestrato (BART-base).

La configurazione `n_docs` definisce il numero di *top-k* documenti che il retriever deve recuperare dal suo dataset indicizzato. Difatti, quando viene istanziato il retriever, si passano le configurazioni del modello in modo che sia consapevole delle modifiche che avvengono in modo dinamico alle configurazioni `model.config.passages_path` e `model.config.index_path`.

Infine, come anticipato nella sezione 4.4, l'output del retriever, quindi l'input del generatore, può avere dimensioni (`max_combined_length` diverse a seconda della soluzione adottata.

```
model = RagSequenceForGeneration.from_pretrained_question_encoder_generator(
    "facebook/dpr-question_encoder-single-nq-base", args.checkpoint_generator).to(device)

model.config.n_docs = 5

retriever = RagRetriever(model.config, question_encoder_tokenizer, generator_tokenizer)

if args.sentence_mode:
    retriever.config.max_combined_length = 768
else:
    retriever.config.max_combined_length = 1024

model.set_retriever(retriever)
```

La fase di addestramento del modello è definita nel metodo `train()`. All'inizio viene istanziato l'*optimizer* [3] con un learning rate [4] uguale a $1e-6$ per tutti gli esperimenti.

Ad ogni epoca, e per ogni input, viene modificato il dataset in cui il componente DPR del modello cercherà i 5 chunk più rilevanti per l'input.

Infine, il componente *accelerator* [5] permette di impostare la precisione di calcolo a 16 bit, invece di 32, al fine di utilizzare meno memoria.

```
accelerator = Accelerator(fp16=True)

def train(model):
    optimizer = torch.optim.Adam(params=model.parameters(), lr=args.lr)

    model, optimizer, train_dataset_loader = accelerator.prepare(model, optimizer,
        train_dataset_model_input)

    lr_scheduler = get_linear_schedule_with_warmup(
        optimizer=optimizer,
        num_warmup_steps=len(train_dataset_loader)//10,
        num_training_steps=len(train_dataset_loader) * args.num_epochs
    ) # Instantiate learning rate scheduler

    # Training loop.
    print("Initiating fine-tuning on our dataset")

    progress_bar = tqdm(range(args.num_epochs * len(train_dataset_loader)),
        disable=not accelerator.is_main_process)

    model.train() # Set the model in the training mode.
    for epoch in range(args.num_epochs):
        # The data loader passes data to the model based on the batch size.
        for i , data in enumerate(train_dataset_loader, 0):
```

[3] *Optimizer*: algoritmo che aggiorna i parametri del modello al fine di ridurre la loss.

[4] *Learning rate*: iperparametro che definisce la velocità di apprendimento.

[5] *Accelerate*: modulo realizzato per gli utenti di PyTorch che preferiscono riscrivere il *training loop* e, allo stesso tempo, utilizzare un'API che permetta di sfruttare più GPU/TPU o precisione di calcolo a 16 bit (fp16).

```
25              # Create the indexed dataset.
26              create_and_set_indexed_dataset(train_dataset_retriever_list[i])
27              # Create tensors to pass to the model
28              input_ids = torch.tensor(data["input_ids"], device=device).unsqueeze(0)
29              attention_mask = torch.tensor(data["attention_mask"], device=device).unsqueeze(0)
30              labels = torch.tensor(data["labels"], device=device).unsqueeze(0)
31
32              # The model outputs the first element that gives the loss for the forward pass.
33              outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
34              loss = outputs[0]
35
36              if i % len(train_dataset_loader) == 0:
37                  print(f"Epoch: {epoch}, Loss: {loss.item()}")
38
39              # Backpropagation.
40              accelerator.backward(loss)
41              optimizer.step()
42              lr_scheduler.step()
43              optimizer.zero_grad()
44              progress_bar.update(1)
45
46          model.save_pretrained(model_path) # Save the model at the end of each epoch.
47          if args.eval_with_bart:
48              model.generator.save_pretrained(model_generator_path)
```

4.5.5 Evalutation

Una volta istanziato il test set (riga 36), si ha la possibilità di valutare il modello generando predizioni con RAG (DPR + BART) oppure solo col generatore (BART): nel primo caso sarà necessario creare la lista dei "dataset" per il retriever (riga 42). Il metodo `generate_predictions()` ha il compito di richiamare il modello per generare il riassunto di ogni chunk nel test set. Il metodo `concatenate_summaries()` si occuperà di concatenare le predizioni facenti parte dello stesso documento.

```
1   def generate_predictions(batch):
2       if not args.eval_with_bart:
3           # Create dataset indexed
4           create_and_set_indexed_dataset(eval_dataset_retriever_list[batch["chunk_id"]])
5
6       inputs = tokenizer(batch["text"], padding="max_length", max_length=max_len_source,
7           return_tensors="pt", truncation=True)
8       input_ids = inputs.input_ids.cuda()
9       attention_mask = inputs.attention_mask.cuda()
10      predicted_summary_ids = model.generate(
11          input_ids=input_ids,
12          attention_mask=attention_mask,
13          max_length=max_len_summary,
14          num_beams=2,
15          repetition_penalty=2.5,
16          length_penalty=1.0,
17          early_stopping=True)
18      batch["predicted_summary"] = tokenizer.batch_decode(predicted_summary_ids,
19          skip_special_tokens=True)
20      return batch
21
22  def concatenate_summaries(preds, dataset_idx):
23      final_summaries = []
24      # For each chunk.
25      for i in dataset_idx:
```

```
26          # Build its final summary by concatenating the summaries of its chunks.
27          summary = ""
28          for j in range(i):
29              summary = summary + preds[j][0] + " "
30          final_summaries.append(summary)
31          # Delete the first "i" predictions to compute the next document.
32          del preds[:i]
33      return final_summaries
34
35  eval_dataset_chunked = Dataset.from_pandas(test_dataset_chunked)
36  eval_dataset_chunked = eval_dataset_chunked.rename_column("Unnamed: 0", "chunk_id")
37
38  if args.eval_with_bart:
39      tokenizer = generator_tokenizer
40      model = BartForConditionalGeneration.from_pretrained(model_generator_path).to(device)
41  else:
42      eval_dataset_retriever_list = create_dataset_retriever_list(eval_dataset_chunked,
43          test_dataset_chunked, args.sentence_mode, args.random_chunks)
44
45  predictions = eval_dataset_chunked.map(generate_predictions)
46  references = test_dataset["summary"]
47  predictions_final = concatenate_summaries(predictions["predicted_summary"],
48      test_dataset_chunked_idx["idx"])
```

I metodi `get_rouge_metrics()` e `get_redundancy_scores()` restituiscono i valori che misurano la bontà delle predizioni del modello (vedi sezione 3.3).

```
1   def get_rouge_metrics(preds, refs):
2       rouge_output = rouge.compute(predictions=preds, references=refs, use_stemmer=True)
3       return {
4           "r1": round(rouge_output["rouge1"].mid.fmeasure, 4),
5           "r2": round(rouge_output["rouge2"].mid.fmeasure, 4),
6           "rL": round(rouge_output["rougeL"].mid.fmeasure, 4)
7       }
8
9   def get_redundancy_scores(preds):
10      sum_unigram_ratio = 0
11      sum_bigram_ratio = 0
12      sum_trigram_ratio = 0
13      all_unigram_ratio = []
14      all_bigram_ratio = []
15      all_trigram_ratio = []
16
17      sum_redundancy = 0
18      stop_words = set(stopwords.words("english"))
19      count = CountVectorizer()
20      all_redundancy = []
21
22      number_file = len(preds)
23
24      for p in preds:
25          all_txt = []
26          all_txt.extend(word_tokenize(p.strip()))
27
28          # uniq n-gram ratio
29          all_unigram = list(ngrams(all_txt, 1))
30          uniq_unigram = set(all_unigram)
31          unigram_ratio = len(uniq_unigram) / len(all_unigram)
32          sum_unigram_ratio += unigram_ratio
33
34          all_bigram = list(ngrams(all_txt, 2))
35          uniq_bigram = set(all_bigram)
36          bigram_ratio = len(uniq_bigram) / len(all_bigram)
37          sum_bigram_ratio += bigram_ratio
38
```

```
39          all_trigram = list(ngrams(all_txt, 3))
40          uniq_trigram = set(all_trigram)
41          trigram_ratio = len(uniq_trigram) / len(all_trigram)
42          sum_trigram_ratio += trigram_ratio
43
44          all_unigram_ratio.append(unigram_ratio)
45          all_bigram_ratio.append(bigram_ratio)
46          all_trigram_ratio.append(trigram_ratio)
47
48          # NID score
49          num_word = len(all_txt)
50          all_txt = [w for w in all_txt if not w in stop_words]
51          all_txt = [' '.join(all_txt)]
52
53          x = count.fit_transform(all_txt)
54          bow = x.toarray()[0]
55          max_possible_entropy = np.log(num_word)
56          e = entropy(bow)
57          redundancy = (1-e/max_possible_entropy)
58          sum_redundancy += redundancy
59          all_redundancy.append(redundancy)
60
61      return all_unigram_ratio, all_bigram_ratio, all_trigram_ratio, all_redundancy
```

4.6 Tecnologie utilizzate

Python

Python è linguaggio ad alto livello e multi-paradigma rilasciato per la prima volta nel 1991. Deriva dal miglioramento di un altro linguaggio, chiamato ABC, sviluppato negli anni ottanta da un team di ricerca della National Research Institute for Mathematics and Computer Science (CWI) di Amsterdam.

Python supporta la programmazione object-oriented, strutturata e funzionale. Inoltre, è multi-piattaforma in quanto è un linguaggio interpretato, cioè può essere eseguito su qualsiasi piattaforma che disponga dell'interprete apposito. É semplice da imparare, dispone di una *standard library* molto ampia e di un sistema di *garbage collector*, che si occupa automaticamente dell'allocazione e del rilascio della memoria.

Python viene utilizzato in numerosi ambiti, tra cui programmazione web, programmazione di reti e accesso a database. È particolarmente popolare in ambito Data Science e Artificial Intelligence, viste le molteplici librerie esterne disponibili. Viene utilizzato sia per la preparazione dei dati sia per la creazione e la valutazione dei modelli.

Google Colaboratory

Google Colaboratory (Colab) è una piattaforma gratuita fornita da Google, che permette di scrivere ed eseguire codice Python sul proprio browser, sfruttando macchine virtuali, oltre a risorse hardware in remoto come CPU, GPU e TPU, che permettono di migliorare notevolmente le performance in termini di

velocità. Il tool è basato su Jupyter, ma non necessita di setup o installazioni. É inoltre possibile inserire celle di testo, immagini, LaTex, HTML e molto altro, in modo da migliorare la leggibilità e la comprensione del notebook.

L'utilizzo di Colab permette di salvare su Google Drive i propri notebook per averli sempre a disposizione, di condividerli con altri utenti e di dare la possibilità a quest'ultimi di commentare o modificare.

L'interfaccia si presenta nel seguente modo:

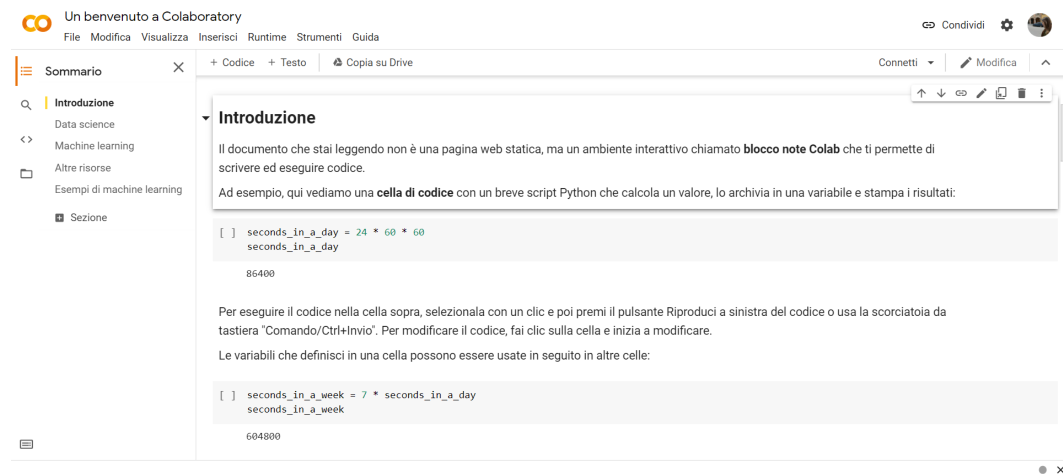


Figura 4.7: Interfaccia della piattaforma Google Colaboratory.

PyTorch

PyTorch [36] è un framework open-source per il machine learning rilasciata nel 2016 dal team Facebook's AI Research lab (FAIR), basato a sua volta sulla libreria Torch. Essa introduce la classe *Tensor*, che permette di operare con array multidimensionali sfruttando l'accelerazione della GPU. Inoltre, introduce numerose librerie a supporto dello sviluppo di applicazioni di Intelligenza Artificiale tra cui:

- *Transformers* di HuggingFace.
- *ParlAI*: una piattaforma per addestrare e valutare modelli per il dialogo (*dialog models*) su vari task.
- *Accelerate*: permette di addestrare e utilizzare modelli su più GPU o TPU, e di impostare la precisione di calcolo a 16 bit (fp16).
- *Optuna*: automatizza la ricerca degli iperparametri.
- *PyTorchVideo*: libreria di deep learning per il *video understanding*.

Hugging Face

Huggin Face è un provider open-source che permette di costruire, addestrare e distribuire modelli per il Natural Language Processing.

Nella sezione “Models” è possibile consultare la documentazione di circa 12k modelli pre-addestrati disponibili per il download e la sperimentazione su un grande numero di task. Essendo anche una community, è possibile effettuare l’upload di un proprio modello per contribuire allo stato dell’arte. Per importare i modelli transformer nel notebook è necessario installare la libreria con il seguente comando:

```
!pip install transformers
```

Nella sezione “Datasets” è disponibile la documentazione dei dataset che è possibile importare dalla libreria datasets nel seguente modo:

```
!pip install datasets
from datasets import load_dataset
dataset = load_dataset("billsum") #esempio
```

Conclusioni e sviluppi futuri

In questa tesi è stato sperimentato un modello retrieval-augmented generativo, basato su Transformers, per il task di Abstractive Summarization su documenti legali molto lunghi. Per permettere il processamento di questo tipo di documenti, si è deciso di dividerli in chunk e di arricchire, attraverso il meccanismo retrieval del modello RAG, la loro rappresentazione, ossia di incorporare ad ognuno di essi del contesto estratto dal resto del documento.

A tale scopo, sono state testate diverse soluzioni che si distinguevano in base al tipo di contesto che veniva aggiunto dal componente retriever: chunk, frasi o chunk composti da frasi randomiche. Le configurazioni migliori sono risultate essere quelle che utilizzavano le frasi o i random chunks, in quanto evitano meglio il problema della ridondanza.

Nel caso fossero disponibili maggiori risorse hardware, sarebbe opportuno testare l'aggiornamento dei parametri del context encoder, addestrandolo sull'intero training set, in modo che impari a produrre una rappresentazione diversa dei documenti nel retriever. Questo accorgimento potrebbe ulteriormente ridurre la ridondanza nei riassunti generati e contribuire ad un aumento dell'accuratezza. Oltre a ciò, si è scoperto che sarebbero opportuni ulteriori studi sul learning rate per permettere al modello di generalizzare meglio su un numero molto grande di esempi di training.

Dal momento che tutti gli esperimenti sono stati realizzati sfruttando l'implementazione della classe `RagSequenceForGeneration`, si potrebbero ripetere gli stessi testando il comportamento della classe `RagTokenForGeneration` che, a differenza della prima, permette di generare un token alla volta. In questo modo si potrebbe generare una sequenza del riassunto in output a partire da più chunk o frasi, diversificando l'influenza che ha il contesto globale sul testo in input e riducendo la ridondanza.

Le soluzioni proposte hanno un margine di miglioramento e, nel caso si trovasse una configurazione ottimale per il task di Abstractive Long Document Summarization, potrebbero essere applicate anche alla Multi-Document Summarization.

Ringraziamenti

Desidero ringraziare i professori incontrati in questi tre anni che, attraverso l'insegnamento, mi hanno trasmesso conoscenze e competenze molto preziose. In particolare, ringrazio il prof. Gianluca Moro per l'attenzione che ha avuto affinché il mio progetto di tesi fosse il più curato e originale possibile. La presenza del dott. Luca Ragazzi è stata fondamentale per districarmi in un campo che conosco da poco e per questo lo ringrazio moltissimo.

Voglio altresì ringraziare i miei compagni di corso per aver reso speciale la vita universitaria e, in particolar modo, voglio menzionare Enrico e Lorenzo che hanno condiviso con me gli ultimi passaggi di questo percorso.

Infine, mi sento infinitamente grata verso Davide, il quale mi ha aiutato a mettermi sui binari giusti per affrontare gli studi e mi ha insegnato a credere nelle mie capacità.

Bibliografia

[1] W. Pitts W. McCulloch. A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, December 1943.

[2] Frank Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 1958.

[3] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[4] Alan Turing. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, October 1950.

[5] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[6] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[7] Quoc V. Le and Tomás Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[8] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly, 2nd edition, 2019.

[9] 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.

[10] Mehdi Allahyari, Seyed Amin Pouriyeh, Mehdi Assefi, Saeid Safaei, Elizabeth D. Trippe, Juan B. Gutierrez, and Krys J. Kochut. Text summarization techniques: A brief survey. *CoRR*, abs/1707.02268, 2017.

[11] H. P. Luhn. The automatic creation of literature abstracts. *IBM Journal of Research and Development*, 2(2):159–165, 1958.

[12] S. Deerwester, S. Dumais, T. Landauer, G. Furnas, and R. Harshman. Indexing by latent semantic analysis. *J. Am. Soc. Inf. Sci.*, 41:391–407, 1990.

[13] Som Gupta and S. K Gupta. Abstractive summarization: An overview of the state of the art. *Expert Systems with Applications*, 121:49–65, 2019.

[14] Princeton University. What is wordnet?

[15] 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.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[17] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[18] 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.

[19] 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.

[20] Yu Yan, Weizhen Qi, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. Prophetnet: Predicting future n-gram for sequence-to-sequence pre-training. *CoRR*, abs/2001.04063, 2020.

[21] Congbo Ma, Wei Emma Zhang, Mingyu Guo, Hu Wang, and Quan Z. Sheng. Multi-document summarization via deep learning techniques: A survey. *CoRR*, abs/2011.04843, 2020.

[22] Wen Xiao and Giuseppe Carenini. Systematically exploring redundancy reduction in summarizing long documents. *CoRR*, abs/2012.00052, 2020.

[23] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020.

[24] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *CoRR*, abs/2001.04451, 2020.

[25] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. *CoRR*, abs/2007.14062, 2020.

[26] Mikhail S. Burtsev and Grigory V. Sapunov. Memory transformer. *CoRR*, abs/2006.11527, 2020.

[27] Qingyang Wu, Zhenzhong Lan, Jing Gu, and Zhou Yu. Memformer: The memory-augmented transformer. *CoRR*, abs/2010.06891, 2020.

[28] Peng Cui and Le Hu. Sliding selector network with dynamic memory for extractive summarization of long documents. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5881–5891, Online, June 2021. Association for Computational Linguistics.

[29] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. *CoRR*, abs/2005.11401, 2020.

[30] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: retrieval-augmented language model pre-training. *CoRR*, abs/2002.08909, 2020.

[31] Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. Latent retrieval for weakly supervised open domain question answering. *CoRR*, abs/1906.00300, 2019.

[32] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. *CoRR*, abs/2004.04906, 2020.

[33] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

[34] Anastassia Kornilova and Vlad Eidelman. Billsum: A corpus for automatic summarization of us legislation, 2019.

[35] HuggingFace. Rag - overview.

[36] PyTorch.org. Ecosystem tools.