POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Informatica

Tesi di Laurea Magistrale

Progettazione e sviluppo di un sistema di generazione automatica dei testi partendo da dati strutturati

Relatore/i

prof. Alessandro Fiori

Candidato

Fortunato Sabato Sole

Anno Accademico 2020-2021

Per aspera ad astra

Ringraziamenti

Ringrazio innanzitutto il mio relatore Alessandro Fiori per avermi seguito durante il progetto di tesi, ed avermi aiutato a superare tutte le difficoltà che ho trovato. Ringrazio la mia famiglia per tutto il supporto economico ricevuto, per avermi permesso di studiare a Torino finanziandomi gli studi. Ringrazio mio fratello, mio mentore, con cui ho trascorso molti momenti insieme tra Milano e Torino. Ringrazio Andrea, Miriam, Angelo, Gennaro e tutti gli altri ragazzi che ho avuto il piacere di conoscere e di poter trascorrere del tempo a Torino, tra le varie bevute a San Salvario e le nottate passate a studiare per consegnare dei progetti universitari. Ringrazio tutti coloro mi sono stati vicini in questi anni, e soprattutto ringrazio me stesso, per non aver mai mollato neanche di fronte alle avversità e per avercela messa tutta per arrivare dove sono ora.

Sommario

Capitolo 1

Introduzione

Il progresso tecnologico, e soprattutto la creazione di calcolatori sempre più complessi e più potenti hanno stravolto totalmente il nostro mondo, entrando a farne parte, e spingendo l'uomo sia ad interagire sia ad essere sempre più spesso dipendente dai computer.

Il machine learning soprattutto ha avuto il compito di riuscire ad inserirsi in tutte quelle attività che storicamente erano sempre collegate agli esseri umani, come per esempio automobili a guida autonoma oppure la generazione di testo in maniera automatica. Infatti, il processo di generazione del testo presuppone che ci sia dietro una mente che sia in grado di pensare ed elaborare e scrivere un testo che sia coerente, e soprattutto che rispetti determinate specifiche.

Le reti neurali sono utilizzate da poco tempo, soprattutto se si pensa da quanto tempo esistono i computer. Negli ultimi venti anni sono stati fatti passi da gigante nell'ambito dell'intelligenza artificiale, riuscendo ad implementare sia da un punto di vista teorico sia pratico dei modelli di rete che fossero in grado di sfruttare a pieno tutta la potenza computazionale disponibile in un determinato periodo storico. Infatti, il collo di bottiglia è legato soprattutto all'hardware, dato che le reti fanno un uso massiccio di CPU e RAM per effettuare calcoli sempre più complessi. Inoltre, computer più potenti permettono di spingere sulle architetture, per fare dei modelli sempre più profondi e potenti. Da come si può notare esso non è affatto un processo semplice, e nel corso degli ultimi decenni sempre più ricercatori si stanno interessando al seguente ambito, sia da un punto di vista teorico sia pratico.

L'obiettivo del seguente elaborato di tesi è quello di riuscire ad implementare un'applicazione che sia in grado di generare del testo in linguaggio naturale a partire da dati strutturati. I dati strutturati saranno dei dati appartenenti a determinate categorie, nel nostro caso particolare essi saranno relativi a determinate categorie di prodotti tecnologici ("stampanti" e "smart-tv"). L'obiettivo non è quello di sviluppare una nuova architettura di rete, dal momento che non è semplice implementare da zero una rete neurale, e soprattutto che essa abbia delle prestazioni superiori a quelle attualmente in circolazione.

Il seguente elaborato è stato suddiviso in vari capitoli, i quali tratteranno aspetti diversi del progetto, in particolare:

- Nei primi due capitoli ci si concentra dapprima sul NLG in generale da un punto di vista teorico, architetture di rete usate nell'ambito del Natural Language Generation, ed infine lo stato dell'arte attuale legato alla generazione di testo,

concentrandosi su eventuali soluzioni sia open source sia commerciali attualmente disponibili.

- Nel capitolo 4 ci si concentrerà sulle varie tecnologie utilizzate all'interno del seguente progetto.
- Nel capitolo 5 sarà descritta l'architettura dell'applicazione, la quale sarà divisa in vari moduli, in cui il modulo principale è quello che coinvolge la rete neurale. Saranno descritte tutte le varie scelte architetturali che ci sono state, ed il perché sia stata fatta una determinata scelta.
- Nel capitolo 6 si parlerà della pipeline, quindi come è stato implementato il progetto da un punto di vista pratico, cercando di proporre quando possibile dei frammenti di codice sorgente e degli esempi legati al flusso di dati lungo la pipeline.
- Nel capitolo 7 saranno descritti tutti i risultati che sono stati ottenuti, cercando di valutarli in maniera sia empirica sia manualmente.
- Nell'ultimo capitolo, invece, saranno presentate le conclusioni ed eventuali sviluppi per lavori futuri.

Capitolo 2

NLG

Il Natural Language Generation (NLG) è una sottocategoria del NLP, con cui si riesce a generare del testo in maniera automatica a partire da dati strutturati, alla velocità di migliaia di pagine al secondo. Il progresso tecnologico ci spinge maggiormente verso l'automatizzazione di tutti quei processi che non potrebbero essere automatizzabili, come nel caso della generazione di testo. Infatti, *(Wigmore, 2021)* l'obiettivo di tutti i vari studi e ricerche è proprio quello di riuscire a generare del testo che sembri esser scritto da un essere umano e che, inoltre, sia corretto da un punto di vista sintattico e semantico.

La generazione del testo deve tener conto di vari aspetti del linguaggio, tra cui la struttura, grammatica, ed un lessico adeguato al contesto. L'obiettivo dell'intelligenza artificiale è da sempre quello di riuscire a semplificare la vita delle persone, riuscendo a ridurre i task che devono essere necessariamente svolti da un essere umano. Fin da sempre, infatti, generare del testo in maniera automatica poteva essere fatto soltanto dagli esseri umani, ma grazie al progresso tecnologico ed allo sviluppo dell'AI, oggigiorno siamo in grado di generare dei testi sempre più complessi e realistici grazie all'uso di un semplice computer.

L'impatto maggiore legato all'uso di una intelligenza artificiale per tali task è legato soprattutto al risparmio di tempo, dato che, per esempio, non è più necessario un operatore per generare un report di tipo finanziario a partire da dei semplici dati, oppure per generare delle recensioni di prodotti a partire dalle loro specifiche.

2.1 Applicazioni NLG

Negli ultimi anni sono state fatte molte ricerche, e soprattutto le aziende si stanno interessando sempre più al problema dell'intelligenza artificiale, la quale viene usata in diversi ambiti, e soprattutto per diversi scopi.

Un sistema NLG può essere usato per tanti scopi, uno fra i tanti può essere quello di una chatbot che oggigiorno è molto utilizzato dalla maggior parte delle aziende, ed ha il ruolo di un operatore che è in grado di capire il contenuto delle domande e dare delle risposte che siano coerenti. Il vantaggio principale legato all'uso di una chatbot è la presenza di un operatore sempre disponibile.

Sullo stesso principio del NLU ("Natural Language Understanding") si basa l'assistente vocale, il quale è in grado di convertire dei suoni in testo, e a partire da quel testo elaborarlo ed eseguire determinate operazioni. Tra gli assistenti vocali più famosi ed intelligenti non si può non citare Alexa oppure Siri, intelligenze artificiali sviluppate rispettivamente da Amazon ed Apple.

Inoltre, con le tecnologie attuali si sta riuscendo a sviluppare dei software che siano in grado di effettuare diverse tipologie di task, come per esempio la traduzione automatica di un testo da una lingua ad un'altra, come nel caso di pagine web in lingue straniere che vengono tradotte in maniera automatica.

Oltre a ciò, vi è il cosiddetto ambito della *(Joshi, 2021)* content creation, ovvero generazione di contenuto in maniera automatica. Per quanto riguarda la content creation vi possono essere alcuni casi differenti: un caso può essere la generazione di testo a partire da dei dati strutturati, per esempio in formato JSON, oppure da grafici, con la generazione di un vero e proprio report. Un altro caso può essere quello di genare del testo che sia la continuazione naturale di ciò che gli è stato passato come input, con l'obiettivo di far risultare il passaggio da una frase ad un'altra quanto più naturale possibile, e mai brusco.

Infine, si potrebbero citare ancora programmi che sono in grado di generare dei riassunti di un testo generico, come potrebbe essere magari un libro, oppure suggerimenti per la prossima parola che trova applicazione nell'ambito degli smartphone, in cui data una parola il sistema riesce a prevedere in maniera adattiva quale sarà la prossima parola ad essere selezionata.

Questi sono soltanto alcuni dei possibili ambiti in cui oggi viene utilizzato NLG, ed il numero effettivo di ambiti in cui viene utilizzato cresce a dismisura di anno in anno.

2.2Pipeline

Il processo di generazione di testo in linguaggio naturale è un processo multi-stage in cui ogni step perfeziona i dati utilizzati in precedenza. Gli step, come si evince dalla Figura *2.1*, sono sei, e sono in ordine:

- (Wigmore, 2021)**Content analysis**, tramite cui si effettua un filtraggio dei dati per determinare quali dovranno essere presenti all'interno del documento
- (Wigmore, 2021)**Data understanding**, tramite cui i dati sono interpretati, e questa è la parte in cui girano gli algoritmi di machine learning per il training della rete e per la generazione dei dati di output. In questo step, quindi, viene definito il modello.

- (Wigmore, 2021)**Document structuring**, tramite cui viene definito come deve essere strutturato il documento finale.
- (Wigmore, 2021)**Sentence aggregation**,tramite cui le varie frasi vengono combinate tra loro per poter generare un testo.
- (Wigmore, 2021)**Grammatical structuring**, tramite cui si applica regole grammaticali e sintattiche tali da rendere il risultato finale quanto più naturale possibile.
- (Wigmore, 2021)**Language presentation**, attraverso cui si trova un modo per rappresentare visivamente l'output, formattando il testo in modo accurato.

Figura 2.1 Schematizzazione step NLG

Nell'ambito della generazione di testo definire una pipeline è il punto da cui partire per poter ottenere l'output desiderato. I seguenti sei step possono esser visti come una metodologia da dover applicare per poter ottenere un risultato soddisfacente, indipendentemente dalle scelte architetturali che si vuole adottare.

2.3Architetture

Nel corso degli anni sono state definite molteplici architetture, ognuna delle quali è basata su un diverso modello, e soprattutto ognuna di essa avrà delle prestazioni diverse. Esse sono:

- Catena di Markov
- RNN
- LSTM
- Transformers

Come è stato già detto, ognuna di esse presenta una struttura differente, e dei pro e contro. Vediamole nel dettaglio.

2.3.1 Catena di Markov

La **catena di Markov** è stato il primo algoritmo applicato nell'ambito della generazione del testo, ed in generale esso è l'algoritmo più semplice da applicare. In essa si usa un processo casuale e senza memoria, ed è basato sulla teoria dei grafi in cui sono presenti stati e transizioni. Dati degli stati, vi saranno degli archi che permettono il passaggio da uno stato ad un altro con una probabilità p, quindi una catena di Markov ci mostrerà tutte le possibili transizioni tra questi stati e la probabilità con cui ognuno di essi accada. Un esempio è rappresentato nella Figura 2.2, laddove si è cercato di schematizzare stati e transizioni per aumentare l'espressività del seguente concetto.

Essa può essere facilmente applicabile nell'ambito dell'NLG poiché il modo in cui una frase viene strutturata può essere facilmente modellato tramite una catena markoviana, per cui ogni parola può esser vista come uno stato, e perciò partendo da quella parola sarà possibile raggiungere con una probabilità diversa tante altre parole. Per cui, verrà scelta come parola successiva, quindi stato successivo, la parola a probabilità maggiore.

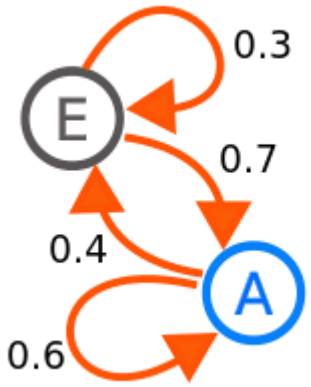

Figura 2.2 Un esempio di catena markoviana con la presenza di stati e transizioni

Data la natura del problema, se ne evince subito che tale modello è di semplice implementazione e non richiede molte risorse computazionali, ma ha lo svantaggio di non essere molto accurato, dal momento che riesce a prevedere soltanto la probabilità della prossima parola che deve essere estratta; quindi, non mantiene nessuna traccia né del contesto né di coerenza del testo.

2.3.2 RNN

Con il termine RNN si definisce un tipo di rete neurale dove l'output proveniente dal layer precedente viene passato come input allo step corrente. A differenza delle reti tradizionali, laddove tutti gli input e output sono indipendenti l'uno dall'altro, nell'ambito della generazione di testo nel caso in cui si deve prevedere la parola successiva di una frase, allora è necessario ricordare le parole precedenti.

La Figura 2.3 ben schematizza qual è il significato di rete Ricorrente, laddove l'output di un generico step diventerà input, tramite le Recurrent connections. La

caratteristica principale di una RNN è l'hidden state, il cui obiettivo è quello di effettuare una serie di calcoli per poter determinare qual è la distribuzione di probabilità della prossima parola, e tener traccia internamente di alcune informazioni su una sequenza. Tali strati nascosti dovranno essere caratterizzati da un determinato numero di parametri, a cui a sua volta saranno associati determinati pesi e bias.

Ma, dal momento che una rete ricorrente può essere vista come un loop che continua ad iterare fino a quando non si sarà raggiunta la fine della frase oppure il carattere di End Of Text, si evince che i pesi e le distorsioni associati a tali celle sono gli stessi. Ciò riduce la complessità dei parametri, a differenza di altre reti neurali.

Alla fine del processo di generazione della prossima parola, verranno calcolate le probabilità di tutte le parole presenti nel dizionario per selezionare quella con probabilità maggiore.

L' algoritmo risulta essere molto più affidabile rispetto alla catena Markoviana, dato la possibilità di memorizzare lo stato attuale del sistema, ma, tuttavia, soffre di alcune limitazioni: non è possibile memorizzare le parole che sono state memorizzate in remoto; pertanto, non va bene nel caso in cui si ha a che fare con frasi lunghe, ma in tal caso ciò che tale algoritmo andrà a fare è quello di effettuare semplicemente una stima sulle parole più recenti che sono state incontrate.

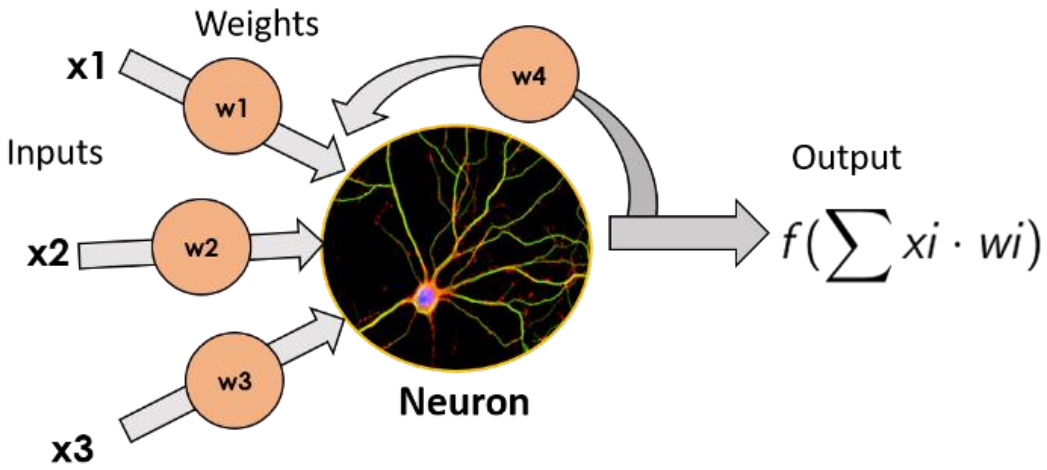

Figura 2.3 Schema ad alto livello di una rete neurale ricorrente

2.3.3 LSTM

LSTM risolve il problema della dipendenza a lungo raggio, ovvero la limitazione della lunghezza delle frasi imposta dalla RNN. LSTM (Long Short-Term Memory) risulta essere una ottimizzazione del RNN, ed è composto, così come riportato in Figura 2.4, da quattro parti:

- Unit
- Input door
- Output door

- Forgotten door

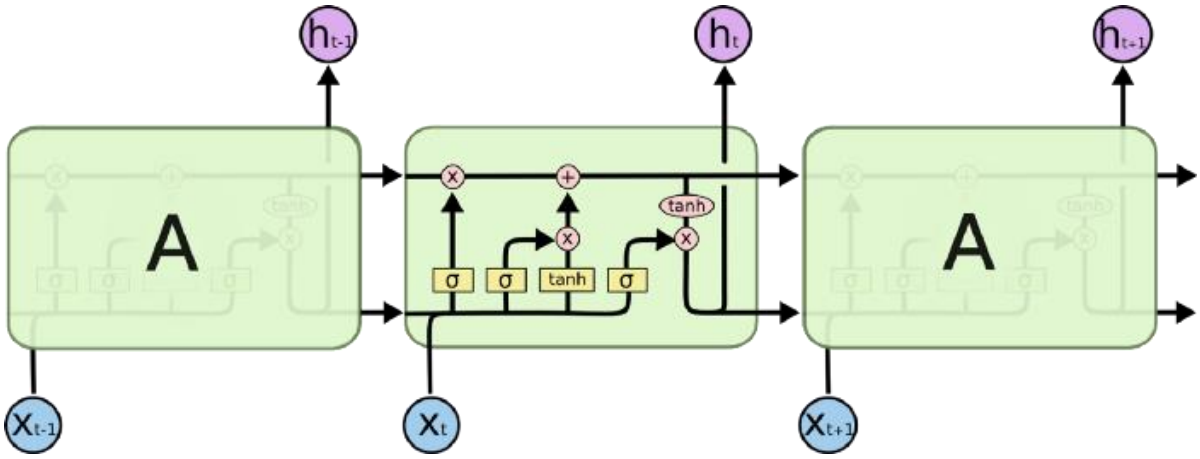

Figura 2.4 Struttura di una cella di una rete LSTM

Ogni cella prende come input la parola i-esima, lo stato della cella precedente e l'output della cella precedente. Manipolando questi input verrà generato un nuovo stato della cella ed un output. Tutte le reti ricorrenti hanno una struttura a catena, quindi la stessa cella verrà ripetuta in loop N volte.

In ogni passaggio temporale, la cella può decidere cosa fare con il vettore di stato: leggere da esso, scrivere su di esso oppure eliminarlo, grazie al meccanismo di gating esplicito. Con input gate, la cella può decidere se aggiornare o meno lo stato della cella. Con il forgotten gate la cella può cancellare lo stato della sua memoria, e con il gate di uscita la cella può decidere se rendere disponibili o meno le informazioni di output. Questi elementi consentono alla rete di dimenticare o ricordare parole in un intervallo di tempo.

Nel momento in cui viene rilevato un nuovo periodo, il forgotten gate può decidere se modificare oppure no il proprio stato interno, risolvendo in questo modo il problema della memoria. Ciò consente alla rete di tener traccia solo di informazioni rilevanti, e pertanto consente al modello di ricordare informazioni per un periodo più lungo. La capacità di memoria di LSTM non è infinita, ma è la stessa del RNN, con la sola differenza che è possibile memorizzare o dimenticare selettivamente delle parole.

LSTM, quindi, in definitiva ha il vantaggio di risolvere il problema della memoria, e quindi di riuscire a gestire dei periodi molto più lunghi, ma allo stesso tempo avrà lo svantaggio legato alla complessità computazionale maggiore, dato che una rete del genere risulta essere molto difficile da addestrare e parallelizzare.

2.3.4 Transformer

Un transformer è un modello relativamente recente utilizzato nell'ambito della generazione di testo. Essa è una architettura che permette di trasformare una sequenza in un'altra sequenza. Per fare ciò in tale architettura si usano encoder e decoder.

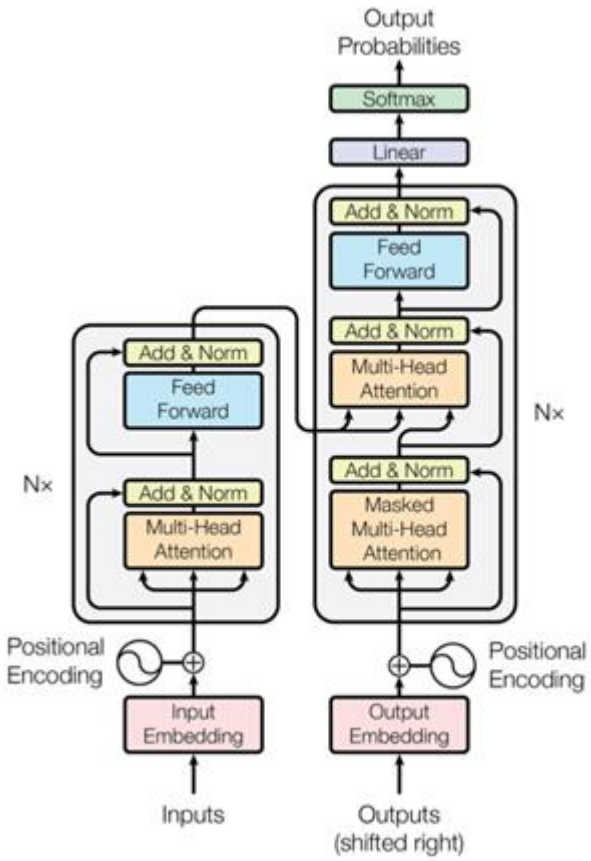

Figura 2.5 Architettura del modello Transformer

Nella Figura 2.5, l'encoder è rappresentato a sinistra mentre il decoder a destra. Sia encoder sia decoder sono impilabili gli uni sugli altri, con un numero di layer N=6. Sia gli ingressi sia le uscite devono prima essere integrati in uno spazio n-dimensionale, dato che non è possibile usare direttamente stringhe, le quali prima di poter essere usate devono essere tradotte in forma vettoriale, con una dimensione di 512.

Tutti gli encoder ricevono, inoltre, come parametro un elenco di vettori, che punta ai vettori delle stringhe. La dimensione di questo elenco corrisponde alla lunghezza massima della frase che può essere generata. Per ogni vettore verranno generati un vettore query, un vettore key ed uno valore, i cui valori saranno generati moltiplicando tale vettore per tre matrici, i cui valori saranno stati definiti durante la fase di training.

Dopo aver definito tali vettori, il secondo passo è composto dal calcolo di uno score. Tale punteggio verrà calcolato prendendo il vettore scalare del vettore query ed il vettore chiave della parola che si sta cercando di valutare. I passi successivi saranno il

dividere gli score per la radice quadrata della dimensione dei vettori chiave e successivamente si normalizzerà tale valore, in modo tale che esso sia compreso tra 0 ed 1. Lo step successivo consiste nel moltiplicare il vettore value per lo score softmax. In questo modo verranno mantenuti i valori delle parole su cui vogliamo concentrarci, mentre verranno eliminate quelle parole che sono ininfluenti.

La (Polosukhin, 2017)Multi-Head attention permette di migliorare le performance dell'attenzione in due modi:

1) espande la capacità del modello di potersi concentrare su posizioni diverse, senza che lo score ottenuto dal vettore sia influenzato dalla parola attuale.
2) fornisce al livello di attenzione più sottospazi di rappresentazione. Non abbiamo più solo uno, ma più set di matrici Query, Key, Value. Ognuno di questi insiemi viene inizializzato in modo casuale.

Lato decoder, invece, oltre agli strati di Multi-Head Attention ed una rete neurale feed-forward, è presente anche un ulteriore strato, Masked Multi-Head attention. Gli strati di self-attention funzionano in modo diverso rispetto all'encoder. Il livello di auto-attenzione è autorizzato ad occuparsi soltanto delle posizioni precedenti nella sequenza di output. Questo viene implementato mascherando le posizioni future.

Lo strato di multi-head attention crea una sua matrice di Query a partire dal livello sottostante, e prende le matrici di Key e Value dall'output dello stack dell'encoder. Lo stack del decoder produce come risultato anch'esso un vettore, che dovrà poi essere trasformato in una parola tramite i layer Linear e Softmax. Lo strato lineare è implementato tramite una rete neurale fully connected, che a partire dal vettore prodotto dai decoder, produce un vettore molto più grande chiamato logits vector. La dimensione di tale vettore corrisponderà al numero di parole presenti all'interno del dizionario di output del modello, ed ogni cella corrisponde al punteggio di una parola.

Lo strato di softmax, infine, trasforma uno score in una probabilità, facendo sì che tutti i valori siano positivi, e che la loro somma sia unitaria.

Ai giorni d'oggi essa è l'architettura dominante nell'ambito dell'NLG. Sono stati definiti vari modelli basati su transformers, come GPT ed altri.

- GPT-2 (A. Radford, 2019) è il precursore del modello GPT-3 ed è stato addestrato su 1.5 miliardi di parametri recuperati da circa 8 milioni di pagine web.
- GPT-3 (T. B. Brown, 2020) è l'ultima versione del Generative Pre-trained Transformer, ed è un modello di linguaggio auto regressivo caratterizzato da circa 175 miliardi di parametri, grazie a cui è in grado di generare del testo molto affidabile, anche senza fare fine tuning.
- BERT, acronimo di "Bidirectional Encoder Representations from Transformers", viene utilizzato per task legati a NLP, presenta una architettura composta da un elevato numero di encoder, dodici nella versione base e 24 nella versione Large. Inoltre, viene usata una rete neurale feed-forward con rispettivamente 768 e 1024 strati nascosti, un numero di attention head di 12 e 16. Il principio di funzionamento della rete è analogo a quello di un generico

transformer (Jacob Devlin), con le sole differenze legate ad una diversa configurazione della rete. Esistono diverse architetture di reti neurali basate su BERT, le quali risultano essere una variante di esso, che differiscono magari per il numero di parametri utilizzati, e quindi per le prestazioni globali della rete.

Capitolo 3

Related Works

Una fase molto importante relativa alla progettazione di un software riguarda lo studio dello stato dell'arte; quindi, cercare se vi sono già dei prodotti commerciali e no, oppure se sono disponibili dei paper da poter usare come punto di partenza oppure come paragone.

Nell'ambito dell'NLP e della generazione del testo è possibile trovare in rete moltissimo materiale e tanti articoli accademici, così come moltissime demo da poter consultare. Nel nostro caso specifico, invece, non è disponibile molto in rete, dal momento che si tratta di un settore di nicchia e di recente espansione.

Il seguente capitolo sarà strutturato nel seguente modo: innanzitutto l'attenzione sarà rivolta ad eventuali software esistenti che si occupano di generazione testuale e poi successivamente ci si concentrerà su specifici modelli di reti neurali da poter utilizzare.

3.1 Kaggle & SimpleNLG

Uno dei punti di forza legati all'utilizzo dell'intelligenza artificiale è legato soprattutto all'ampia community che c'è dietro; quindi, non è molto difficile poter reperire online dei modelli di reti pre-addestrati, oppure informazioni di natura generica. La piattaforma utilizzata per condividere dataset, modelli pre-addestrati oppure codici eseguibili è Kaggle, che attualmente risulta essere la community più attiva nell'ambito del machine Learning.

Quindi, partendo da Kaggle è stato possibile esplorare il mondo della generazione di testo automatica, per poi riuscire a trovare una metodologia per lo sviluppo dell'applicativo. Oltre a ciò, poi, si è cercato se esistessero in commercio dei software che generassero in automatico del contenuto a partire da dati strutturati.

In ambito open source, invece, è presente soltanto (gitHub, s.d.) simpleNLG, la quale è una semplice API scritta in JAVA. Essa viene utilizzata per alcuni progetti di natura accademica o commerciale, e si compone di:

- Un proprio sistema morfologico, in grado di gestire in maniera corretta la morfologia di un linguaggio
- Realizer, con cui generare del testo da una forma sintattica. Nella versione attuale la copertura grammaticale è limitata rispetto ad altri tool.

- Microplanner, che nelle versioni attuali si limita a fare una semplice aggregazione

La versione attuale della release è la V4.5.0, ed è resa disponibile in lingua inglese. Eventuali implementazioni per altri linguaggi sono state fatte da terze parti. È possibile usufruire di tale software a partire dalla repository presente su GitHub.

A livello commerciale sono presenti poche aziende, soprattutto di nicchia, che si occupano di NLG. Nel nostro caso particolare, sono state individuate soltanto due aziende che si occupassero di generazione di testo a partire da dati strutturati: semantika.ai ed **Ax-Semantics**. Nel primo caso non è presente una demo, quindi non è stato possibile verificare come l'applicativo lavora per poter generare del testo, mentre nell'altro caso è presente una demo in cui poter esplorare tutta la soluzione, e vedere come essa funziona.

3.2 Ax-Semantics

L'azienda **Ax-Semantics** è una azienda tedesca che offre servizi di NLG per vari settori. Essa utilizza l'intelligenza artificiale e l'apprendimento automatico per addestrare la sua piattaforma. Inoltre, essa è una delle poche aziende che mette a disposizione una serie di API che è possibile usare per potersi interfacciare con i propri servizi. Quindi, viene introdotto il vantaggio che programmatori di terze parti possono utilizzare servizi provenienti da Ax-Semantics. Per poter generare del testo, per prima cosa dobbiamo caricare i dati strutturati, che sono organizzati in documenti. Il più delle volte un documento rappresenta un singolo oggetto oppure un evento che si vuole descrivere.

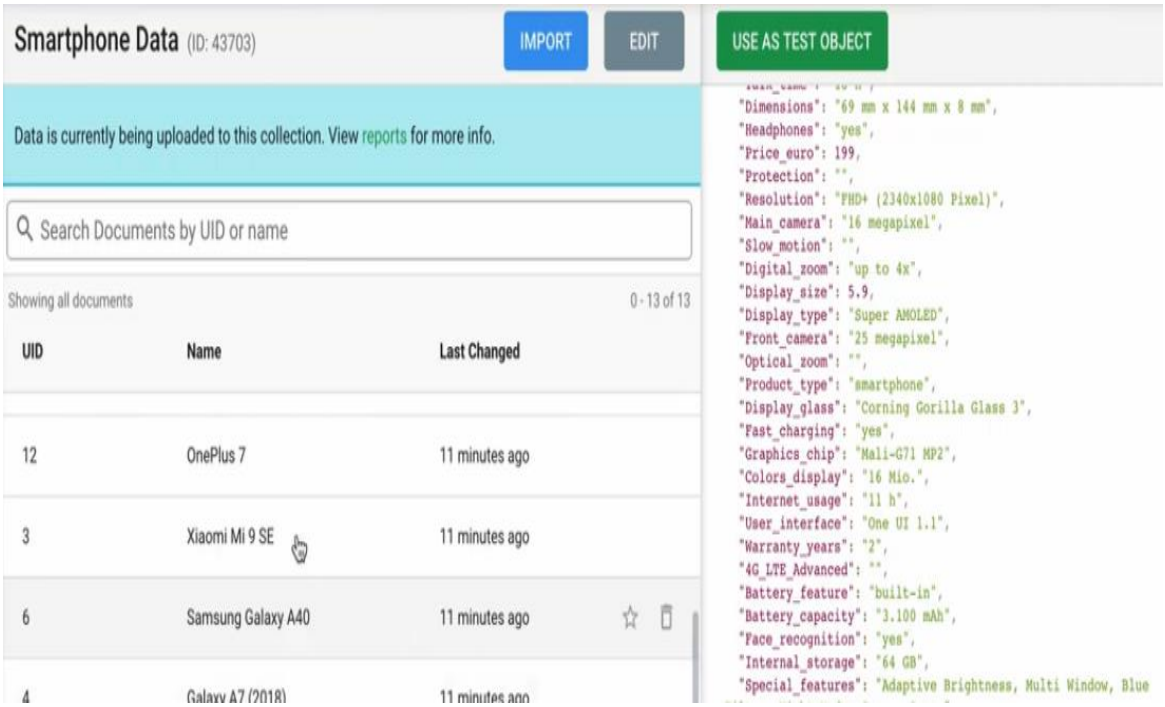

Figura 3.1 Esempio di schermata iniziale di Ax-semantics, da dove è possibile importare i documenti

In Ax-Semantics è possibile caricare dati in diversi formati come JSON, CSV, Excel. È possibile scegliere la lingua del testo che vogliamo generare (Figura 3.1). Cliccando sul bottone generate text, è possibile generare del testo per un singolo documento. Cliccando su generate, viene generato del testo basato su dati passati come input ed un ruleset creato appositamente per quel tipo di dati. Alcune parti di questo testo sono direttamente inserite dagli attributi dei dati, mentre altre parti vengono derivate da essi.

Ogni generazione di testo risulta essere differente, e nel caso di generazioni multiple dello stesso documento il risultato sarà differente. I testi possono variare in:

- Contenuto
- Lunghezza
- Struttura
- Formulazione

Genera una descrizione che sembra una recensione scritta da un essere umano, ed inoltre viene aggiunto anche un elenco dei vari attributi con alcune specifiche tecniche. Il template risulta essere sempre lo stesso, anche se la derivazione del testo è differente.

Talvolta si vorrebbe generare in maniera automatica più di un singolo documento alle volte: in Ax-Semantics è presente un bottone che permette di scegliere tra ALL e filtering nella scelta dei documenti all'interno di una collection per i quali si vuole generare del testo.

Ci sono due modi per usare testi generati:

- Modalità push, con cui i risultati saranno consegnati tramite webhooks dopo ogni generazione
- Pull, tramite cui sarà possibile scaricare i risultati dall'applicazione, oppure tramite varie API

Inoltre, è possibile automatizzare il processo di push completamente e generare automaticamente i testi quanto essi vengono caricati. Viene fornita una rapida generazione di testi di decine di migliaia di testi al minuto, e milioni di testi al giorno. All'interno di tale applicativo sono presenti diversi moduli, ognuno specializzato in una determinata operazione.

3.2.1 Composer

Nel composer l'utente deve costruire un set specifico di regole che saranno usate per generare testo basato sui dati. La costruzione di questo set di regole si riconduce essenzialmente in due parti:

1) Definire logica e condizioni dei dati
2) Determinare come i messaggi devono essere espressi in linguaggio

Per la seconda parte l'utente deve scrivere degli Statement per uno specifico documento, l'oggetto di prova, e la preview mostra in maniera istantanea il risultato.

Tramite il concetto di branching è possibile creare differenti percorsi di testo ed incrementare la varianza del testo. In ogni punto in uno Statement è possibile dividere il testo in branch e sub-branch che saranno in seguito scelte sulla base della logica che è stata scelta dall'utente, oppure vi è la possibilità di dare libera scelta al sistema, che in maniera randomica potrà scegliere uno o un altro percorso, ed in seguito far convergere di nuovo questi diversi percorsi. Più branch si definiscono, più testi differenti è possibile generare.

3.2.2 Container

I container permettono di dare all'utente il focus sugli attributi dei dati. Utilizzando un container il sistema non si limita soltanto ad inserire i dati, ma essi vengono cambiati in modo da rispettare le regole grammaticali del testo. I container hanno molte impostazioni, tra cui è possibile definire:

- Variable input
- Grammatica
- Formattazione di un container

Bisogna soltanto limitarsi a definire la forma delle parti delle frasi. Il metodo più veloce per vedere quali effetti hanno le branch e le configurazioni dei container, è quello di cambiare i test data.

I simboli verdi e rossi presenti all'interno della GUI del container mostrano in quali condizioni delle branch sono prese oppure no. Nella parte inferiore di ogni Statement è possibile vedere l'output del plaintext.

3.2.3 Transformer

Un altro modulo del sistema è il transformer, dove si definisce la logica e le variabili che verranno usate negli Statement. Il tutto viene fatto tramite una interfaccia grafica, così come mostrato in Figura 3.2. Si deve creare dei nodi per diverse funzioni e linkarli tramite differenti porte. Il data node fornisce valori da uno specifico campo. Attraverso le sue porte è possibile passare i valori dei dati ad altri nodi per ulteriori elaborazioni o direttamente agli Statement.

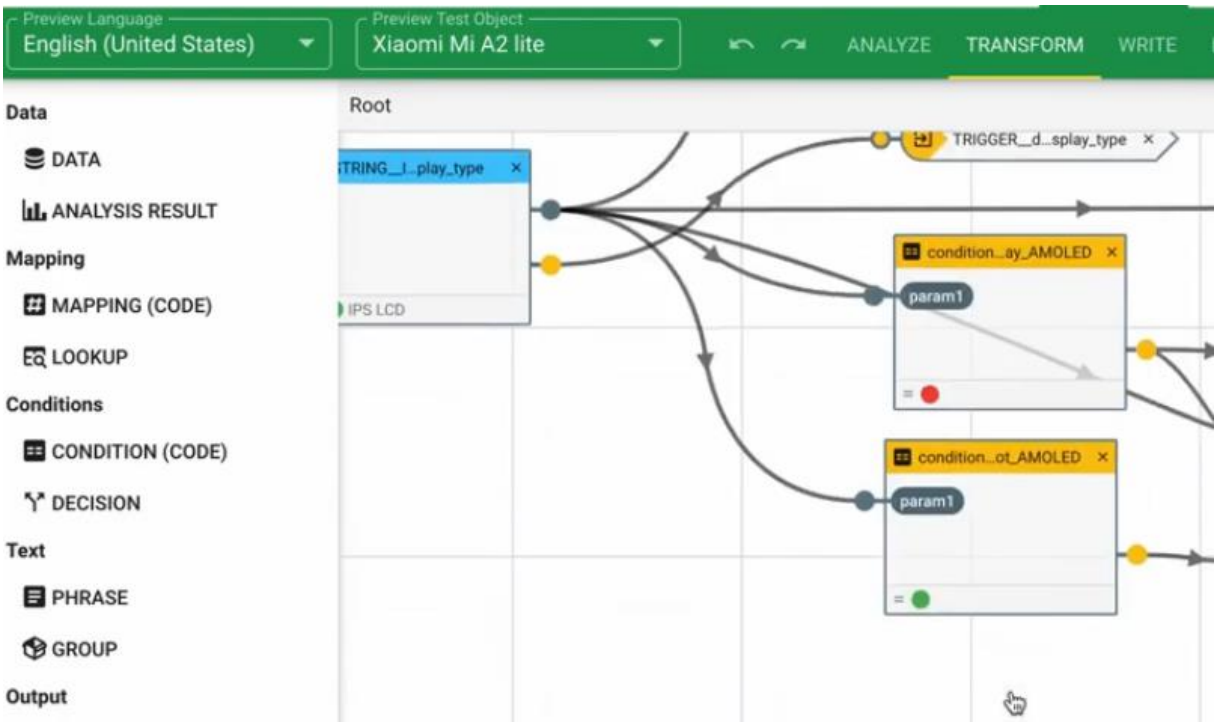

Figura 3.2 Esempio di interfaccia Grafica del modulo Transformer in ax-semantics

3.3 GPT-2

Il modello OpenAI GPT-2 è il successore di GPT, ed è stato teorizzato nell'articolo Language Models are Unsupervised Multitask Learners. Esso è composto da 1,5 miliardi di parametri, ed è stato addestrato su un dataset composto da 8 milioni di pagine.

Tale modello viene utilizzato per task legati alla generazione di testo, e dal momento che il modello è stato pre-addestrato con dati provenienti da vari domini, riesce a generare del testo per molti domini con buoni risultati. GPT-2 è uno scale-up di GPT, con più di 10X parametri ed addestrato su più di 10X dati.

Per il training, GPT-2 usa un approccio di tipo unsupervised. Il modello originale del transformer è composto da una architettura basata su encoder e decoder, ma con i lavori di ricerca più recenti è stato possibile creare delle reti basate o sull'uso di soli encoder oppure decoder, riuscendo ad impilarne il maggior numero possibile, ed addestrando tali reti con enormi quantità di dato.

GPT-2 risulta essere composto da un numero variabile di decoder, il cui numero dipende dalla tipologia di versione che si vuole utilizzare. In particolare, ne esistono quattro: small, medium, large ed extra large, che differiscono soltanto per il numero di decoder presenti, che a sua volta implicherà un numero di parametri differenti, così come mostrato in Figura *3.3*.

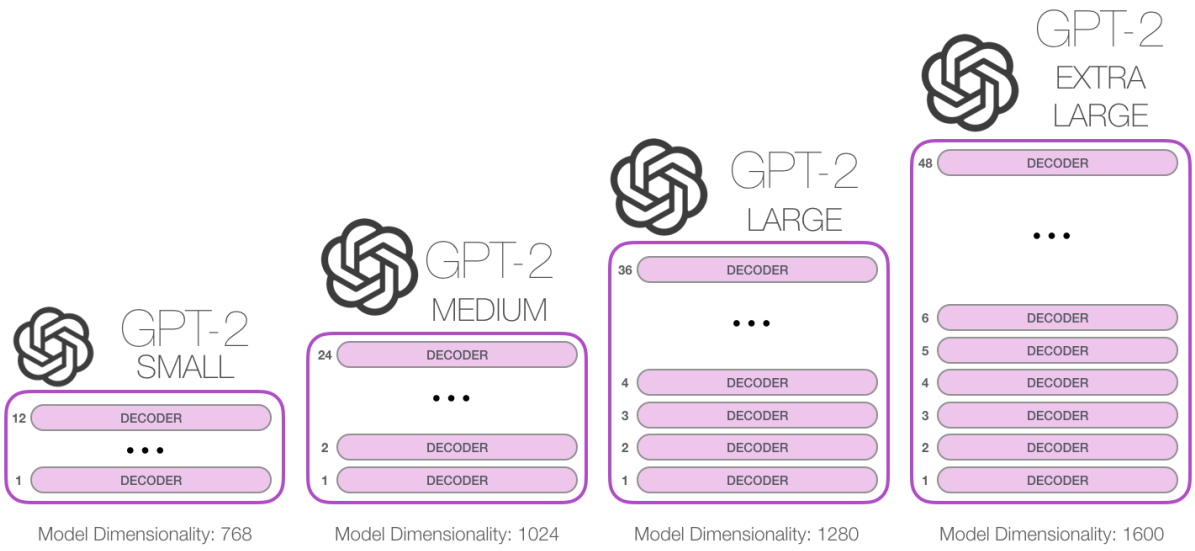

Figura 3.3 Diverse versioni di GPT-2 caratterizzate da un numero diverso di decoder

Il primo step è legato alla codifica dell'input, tramite cui una stringa deve essere convertita in vettore. La dimensione di tale vettore varia a seconda della tipologia di rete GPT-2 che si sta usando, e va, come si evince dalla Figura *3.3*, dai 768 per la small, 1024 per la medium, 1280 nella Large e 1600 nella extra large per ogni parola o token.

Successivamente, dopo avergli sommato la codifica posizionale, viene passato ai decoder. Ogni decoder è composto da uno strato di auto-attenzione e da una rete neurale feed-forward. Quindi, per esempio, una volta che il primo blocco del transformer elabora il token produrrà un vettore risultante che sarà inoltrato al successivo decoder nella rete. Il processo risulta essere identico al precedente, anche se ogni blocco ha i suoi pesi sia nella self-attention sia nella rete neurale.

Quando il blocco superiore nel modello produce il suo vettore di output (il risultato della propria auto-attenzione seguita dalla propria rete neurale), il modello moltiplica quel vettore per la matrice di incorporamento. Dal momento che ogni riga nella matrice di incorporamento corrisponde all'incorporamento di una parola nel vocabolario del modello, il risultato di tale moltiplicazione non è altro che il punteggio di ogni parola nel vocabolario. Si seleziona il token a massimo punteggio oppure altre strategie.

Il modello continua ad iterare fino a quando non saranno generati tutti i token oppure sarà raggiunto il token di end of text. I blocchi di self-attention identificano su quali parole concentrarsi. Tra ogni blocco di auto-attenzione e la rete feed-forward è collocato un layer che fa normalizzazione.

3.3.1 Differenza con BERT

Mentre GPT-2 è implementato usando blocchi decoder, BERT utilizza blocchi encoder. Una differenza fondamentale è che GPT-2 produce un token alla volta. Nel momento in cui viene prodotto un nuovo token, esso viene aggiunto alla sequenza di input, e tale sequenza diventa il nuovo input del sistema. Tuttavia, la dimensione massima di tale sequenza è limitata a 1024 token. GPT-2 inoltre, è un modello di tipo auto-regressivo.

Un'altra differenza la si trova nel layer di self-attention, la quale maschera i token futuri andando a bloccare tutte le informazioni che si trovano a destra della posizione calcolata. La distinzione tra auto-attenzione (che è quella usata da BERT), e masked auto-attention è che: mentre l'auto attenzione consente ad una posizione di raggiungere il picco ai token alla sua destra, in quella mascherata viene impedito che ciò accada.

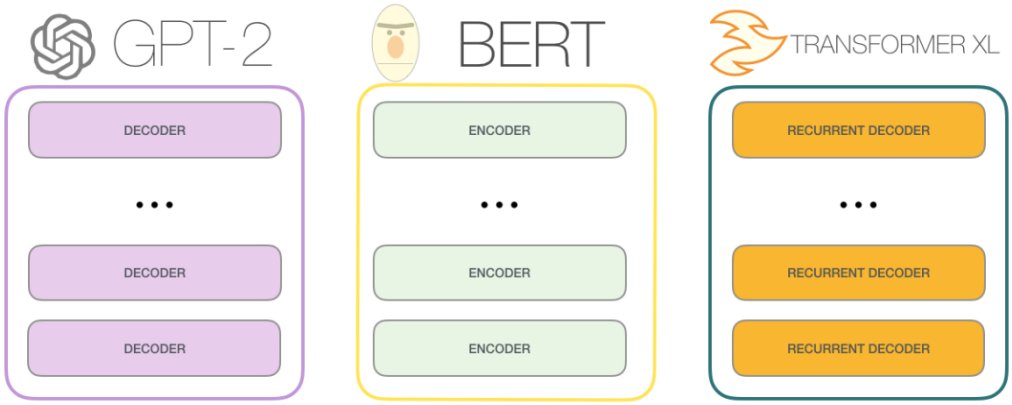

Figura 3.4 Figura che mostra differenza tra GPT-2 basata su uno schema a solo decoder, e rete BERT composta da soli encoder

Anche per quel riguarda le prestazioni, ci sono delle differenze tra le due architetture, anche se dipende molto dal task che si sta eseguendo e dal dataset che si è deciso di utilizzare per l'addestramento della rete. In Figura 3.5 (Jacob Devlin) sono riportati gli score ottenuti su dei task di NLP usando GLUE benchmark, in cui viene confrontato BERT con GPT e se ne valuta le prestazioni attraverso fine tuning differenti e dataset diversi.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$BERT_{BASE}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$BERT_{LARGE}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Figura 3.5 Tabella comparativa prestazioni di diversi modelli di rete ottenuti tramite benchmark GLUE. I risultati sono estratti da BERT: Pre-training of Deep Bidirectional Transformers for

3.4 GPT-3

GPT-3 è un nuovo modello linguistico che è in fase di realizzazione da parte di OpenAI. Esso è stato rilasciato nel 2020 e viene utilizzato per la generazione del testo, anche se non è stato reso disponibile il codice sorgente, ma è utilizzabile tramite opportune API per poter generare del testo a partire dalla rete pre-addestrata.

L'output verrà generato da ciò che il modello ha appreso durante la fase di training. Anche esso è un unsupervised learning, e per effettuare la fase di training è stato usato un dataset contenente circa 300 miliardi di token testuali. GPT-3 genera l'output un token alla volta.

L'architettura è basata su uno schema a decoder, ed in particolare è composta da circa 175miliardi di parametri, contro gli 1,5 miliardi di GPT-2. Tali parametri, che sulla

rete untrained sono inizializzati con valori randomici, costituiscono il modello, e sono utilizzati per calcolare quale token selezionare ad ogni iterazione.

Un ulteriore differenza rispetto a GPT-2 è data da quanti token riesce a gestire in input la rete, che passa da 1024 a 2048. Il numero di decoder all'interno dello stack ha subito una modifica. Tale quantità è stata incrementata a 96 decoder impilati l'uno sull'altro, ed inoltre, ognuno di quei layer è caratterizzato da 1,8 miliardi di parametri usati per fare i calcoli e produrre un output. La differenza è che viene usato uno schema basato su **Sparse Transformer**, che va a sostituire il modello standard basato su Transformers.

(Rewon Child, 2019)Uno Sparse Transformer è un transformer che utilizza fattorizzazioni sparse della "attention matrix" per poter ridurre il rapporto tempo/memoria ad $O(n\sqrt{n})$. Altre modifiche rispetto all'architettura standard sono state: un blocco residuo e l'inizializzazione dei pesi, un insieme di kernel di attenzione sparsi che calcolano sottoinsiemi a partire dalle matrici di attenzione, il ricalcolo dei pesi dell'attenzione durante il passaggio all'indietro per ridurre l'utilizzo della memoria.

Capitolo 4

Tecnologie utilizzate

Dopo aver fatto una breve introduzione sul contesto, iniziamo a scendere più nel dettaglio parlando di come è stato sviluppato, dando particolare rilievo alle tecnologie utilizzate.

Partendo da una rappresentazione a blocchi e con un alto livello di astrazione, è possibile immaginare il tutto come una black box in cui dato un input verrà prodotto un generico output. In input avremo file di tipo JSON, mentre in output verrà prodotto un file al cui interno saranno presenti tutti i documenti che sono stati generati dalla pipeline.

4.1 JSON

Prima di continuare con la trattazione, è meglio spendere qualche parola riguardo lo standard JSON. Esso è un formato utilizzato in ambito web per lo scambio di dati, ed è basato su coppie chiave/valore. In Figura 4.1 è mostrato un esempio di un oggetto JSON, dando risalto alla sintassi utilizzata da tale formato per identificare chiavi e valori, e come viene delimitato un oggetto, tramite "{"per delimitare l'inizio, e "}" per segnalare la sua fine.

Il vantaggio principale legato all'uso del formato JSON è soprattutto la sua leggibilità, dal momento che si cerca di scegliere dei nomi per il campo chiave che siano molto descrittivi, e quindi molto comprensivi per persone che provano ad usare tali dati. Ai giorni d'oggi, esso è diventato lo standard attraverso cui avviene lo scambio di dati tra client e server nell'ambito della programmazione distribuita. L'acronimo JSON sta per JavaScript Object Notation.

```
{
"home": "Html.it",
"link": "http://www.html.it",
"argomento": "Standard del web",
"aree": [
{
"area": "CSS",
"url": "http://css.html.it"
},
{
"area": "Basic",
"url": "http://basic.html.it"
}
]
}
```


Figura 4.1 Esempio documento JSON

Tipicamente gli oggetti JSON presentano una struttura di tipo nested, in cui il valore di un campo chiave è a sua volta un altro oggetto con propri campi chiave valore. Bisogna pertanto prestare attenzione nel momento in cui si vuole cercare di accedere ai vari campi.

Un oggetto JSON, inoltre, presenta un altro vantaggio: possono esservi attributi mancanti all'interno di un oggetto, cosa che per esempio non può verificarsi in altri sistemi, laddove nel caso in cui non siano presenti determinati campi devono essere marcati con valore NULL. Il nostro file di input non è nient'altro che un vettore di JSON Object, che da un punto di vista implementativo è delimitato da parentesi quadre, ed ogni oggetto è delimitato da parentesi graffe, e separato tramite virgola ",".

4.1.1 Dataset

Uno dei primi scogli su cui ci siamo dovuti scontrare è stato quello di reperire un buon numero di dati da poter utilizzare per addestrare la rete. Abbiamo scelto di prendere come corpus un buon numero di reviews di prodotti tecnologici prese dal web.

Il sito web da cui sono state prese tali recensioni è pcmag.com, un sito internazionale molto affidabile, e che soprattutto ha un vasto numero di reviews di prodotti tecnologici. Nel nostro caso le categorie scelte sono state "stampanti" e "smart-tv". Per poter generare tale dataset ci siamo serviti di Scrapy, un web crawler in grado di riuscire a visitare molti collegamenti ipertestuali e salvare i dati di cui abbiamo bisogno.

4.2 Scrapy

Nel concreto, per la generazione di un dataset ci siamo serviti di un (scrapy, 2021) web crawler da poter usare per navigare in maniera automatica attraverso siti web. In particolare, si è scelto di utilizzare **scrapy**, che è un framework disponibile per Python, e con una documentazione molto ricca. Tutto ciò che si è dovuto fare è stato implementare un algoritmo di ricerca in grado di navigare in maniera automatica attraverso i collegamenti ipertestuali, e prelevare da tali pagine il contenuto di cui necessitiamo.
Uno spider è definito come una classe che estende la classe scrapy.Spider, da cui eredita attributi e metodi. Inoltre, deve essere specificato al bot qual è il punto di inizio della navigazione, ed inoltre come seguire i vari link, ed infine come analizzare il contenuto delle pagine scaricate per poter estrarre i dati. Scrapy lavora in maniera asincrona, ovvero non vi è il bisogno di aspettare di ottenere una risposta da parte del server prima di poter proseguire con l'esecuzione.
Per creare un nuovo progetto, è sufficiente usare da shell il comando:

```
scrapy startproject nome_progetto
```

che produrrà come risultato una nuova directory al cui interno saranno generati in maniera automatica diversi file, ognuno di essi aventi un determinato scopo.
In Figura 4.2 viene mostrato un esempio di output ottenuto tramite il comando precedente. In generale, non è necessario editare tali file. Sul sito ufficiale di scrapy vi è presente una guida molto dettagliata, sia su come approcciarsi per la prima volta, con un tutorial che spiega ogni operazione step by step, sia guide più avanzate che permettono di effettuare ottimizzazioni nel codice e nelle performance generali. Nel seguente paragrafo cercherò di presentare gli aspetti base del seguente framework.

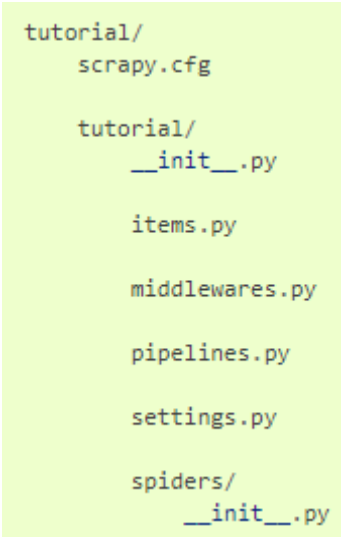

Figura 4.2 Esempio di files generati all'interno di un progetto scrapy

All'interno di un progetto è possibile definire un numero arbitrario di spiders, ognuno caratterizzato da un nome univoco, ed è possibile scegliere a tempo di esecuzione quale si vuole usare.

Per eseguire lo script, da prompt dei comandi si deve accedere alla directory laddove il crawler è stato definito, ed a partire da tale directory digitare il comando:

```
scrapy crawl nome_progetto
```

Tale comando manderà in esecuzione lo spider all'interno della shell, e resterà in esecuzione fino a quando non avrà finito di navigare attraverso i collegamenti ipertestuali. Di base deve essere implementato il metodo start_request(), il quale ha il compito di restituire un oggetto di tipo scrapy.Request, a partire dal quale lo Spider potrà iniziare la fase di scansione.
Nel caso in cui esso non venga esplicitamente implementato, potrà essere sostituito da un vettore *start_urls {nome…}* che verrà utilizzato come parametro per la funzione start_request() nella sua implementazione originaria.
Il metodo parse (), invece, avrà il compito di gestire la risposta generata da ognuna delle richieste, ed eseguire determinate operazioni. Il metodo parse, inoltre può essere usato anche per seguire nuovi collegamenti ipertestuali e creare nuove richieste Request a partire da essi.

Per poter estrarre i dati è possibile seguire due approcci:

1) Utilizzare selettori basati su CSS, usando la sintassi response.css (). get ()
2) Usare selettori basati sulla sintassi xpath, tramite response.xpath().get()

Di base sono disponibili due metodi: get () e getall (). Il primo viene usato nel caso in cui si vuole estrarre soltanto il primo risultato prodotto da un selettore, getall (), invece, permette di estrarre tutto e ritorna pertanto una lista.

Per poter lavorare con i selettori bisogna far pratica con il codice html delle pagine da cui si vuole estrarre dati, ed estrapolare gli id necessari per accedere agli elementi cui si vuole fare accesso.

I selettori xpath, si basano sull'XML per riuscire a selezionare dei nodi all'interno di un albero DOM. Sebbene tipicamente non siano molto usati, tali selettori risultano essere molto più potenti dei selettori CSS, ed inoltre, internamente i selettori CSS verranno convertiti in equivalenti xPath prima di essere utilizzati.

(w3school, 2021)Per poter lavorare con espressioni di tipo xPath vi è una sintassi da dover utilizzare per scrivere delle espressioni di ricerca di nodi. Di seguito in Figura 4.3 verranno riportate alcune espressioni di ricerca di nodi:

Expression	Description
nodename	Selects all nodes with the name "*nodename*"
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

Figura 4.3 Alcuni selettori XPATH

Nel momento in cui viene eseguita una ricerca su un determinato selettore verrà ritornata una lista che potrà contenere zero o più occorrenze che corrispondono a quella desiderata. Pertanto, è possibile iterare e scorrere tale lista, così da poter accedere ad ognuno di essi ed eseguire determinate azioni.

Per generare un file di output, senza dover dichiarare in maniera esplicita l'operazione di scrittura su file, è possibile usare il seguente comando da shell:

```
scrapy crawl nome_spider -O nome_file_output.json
```

In questo modo, è possibile generare un file di output, che nel caso dell'esempio sopra riportato produrrà come output un file JSON contenente tutti gli elementi estratti dallo spider. Utilizzando il comando -O si specificherà che qualora vi sia già un file con il medesimo nome, esso sarà rimpiazzato e sostituito dal nuovo file.

4.2.1 Caso pratico

Nel nostro caso, sono stati implementati i metodi parse(self,response) e parse_item(self,response). Il compito del metodo parse è quello di navigare attraverso

i vari collegamenti ipertestuali e richiamare la funzione parse_item sul response ottenuto dalla pagina i-esima.

Il metodo parse_item non farà altro che iterare sul response della pagina HTML, ed utilizzando la sintassi xpath estrarrà il testo soltanto dagli elementi contenuti all'interno del tag <article></article>, ed in particolare si preleverà tutto il testo contenuto nei tag discendenti dal tag precedente.

Quindi, il metodo parse () si occuperà di scorrere la lista di tutti i collegamenti ipertestuali e generare delle nuove scrapy.Request() che avranno come callback self.parse_item(). Il metodo parse_item(), invece, si occuperà di estrarre determinati pattern dal response.

Verrà prelevato il testo da ogni tag paragrafo nell'ordine in cui essi compaiono nel testo, e scritti su file. Dal momento che un documento HTML è costituito da un insieme di paragrafi, il risultato sarà una lista di stringhe separate da una virgola e delimitate da doppi apici. Pertanto, tali stringhe devono essere pre-processate.

Per quel che riguarda il pre-processing di tali stringhe, l'idea che c'è dietro l'algoritmo è molto semplice, e si basa sul fatto che ogni oggetto JSON a sua volta è costituito da una lista di stringhe. Pertanto, è bastato iterare su tale lista e concatenare tutte le varie stringhe in un'unica stringa. Dopo aver iterato su tutto il vettore di oggetti JSON, viene generato in output un nuovo file serializzato in formato JSON

```
with open('nome_file.json, 'w', encoding='utf-8') as f:
    json.dump(data, f, ensure_ascii=False, indent=4)
```

Tabella 4.1 Istruzione Python che effettua scrittura su file in formato JSON

con le seguenti righe di codice si apre uno stream in scrittura, usando come codifica UTF-8, così da fornire una codifica ad eventuali caratteri speciali presenti all'interno delle stringhe che non rientrano nella codifica ASCII. Infine, con json.dump si serializza il tutto in formato JSON, e il tutto viene scritto su file.

Il file prodotto alla fine di questo step farà parte del corpus della nostra rete, e sarà utilizzato per addestrare la rete.

4.3 Ambiente di Sviluppo

Come ambiente di sviluppo è stato scelto Google Colaboratory (denominato Colab). Colab permette di eseguire codice python online, ed è molto usato nell'ambito di intelligenza artificiale, dal momento che viene messo a disposizione delle persone un server dotato di una determinata quantità di RAM, di una determinata CPU ed inoltre di una GPU, che nell'ambito delle reti neurali ricopre un ruolo fondamentale. Di base Colab è gratuito, anche se con determinate limitazioni. Indipendentemente dalle limitazioni legate ad un account free oppure premium, tramite Colab è possibile definire dei blocchi note composti da celle di codice e da celle di testo.

Nell'ambito del machine learning esso è molto utilizzato in quanto viene resa disponibile la funzionalità di condividere del codice oppure degli esempi di codice, tali da poter essere visibili e comprensibili. Inoltre, anche importare un progetto risulta essere molto più semplice.

Nel nostro caso è stata usata una versione gratuita, che è stata integrata nell'ecosistema Google tramite Google Drive (in Figura 4.4 viene mostrato schematicamente come i vari servizi di Google si integrano all'interno di Colaboratory). Infatti, tutti i file caricati oppure generati durante un Runtime verranno automaticamente cancellati dal sistema nel momento in cui la sessione termina. Se invece si effettua un mount di Drive, allora qualsiasi file verrà generato durante l'esecuzione sarà salvato in automatico sul cloud, evitando magari perdite di dati dovute ad inattività. Vi è, inoltre, la possibilità di poter usare al suo interno molte librerie Python, tra cui nell'ambito del ML le più utilizzate sono TensorFlow, Keras, e tante altre.

È possibile installare nuovi componenti all'interno della macchina virtuale utilizzando il comando "pip install" seguito dal nome del package che si vuole installare. Digitando tale comando verrà installata la libreria e tutte le sue dipendenze, ed essa sarà mantenuta in memoria fino allo scadere della sessione.

La massima quantità di memoria utilizzabile, così come sia la quantità di GPU e TPU utilizzabile, e la massima durata di un task in modalità batch risultano essere limitati in modalità free. La memoria, per esempio, è limitata a 12GB, mentre la massima durata di un task è settata a dodici ore. Dopo tale intervallo di tempo la macchina virtuale verrà automaticamente stoppata e la sessione chiusa. Utilizzando una versione premium, invece, tali limiti vengono meno, riuscendo ad accedere ad una quantità maggiore di risorse, con prestazioni maggiori e con task dalla durata maggiore rispetto a quella precedente.

Per riuscire ad offrire un servizio a quante più persone possibili, Google ha implementato sulla sua piattaforma dei meccanismi automatici di logout che fanno disconnettere automaticamente gli users che risultano essere inattivi per più di 30 minuti. Per inattivo si intende che la persona non è fisicamente vicino colab. Tale vincolo, tuttavia risulta essere facilmente aggirabile, scrivendo per esempio un qualche tipo di script che clicchi in ogni determinato intervallo di tempo su una cella vuota, in modo da evitare la disconnessione.

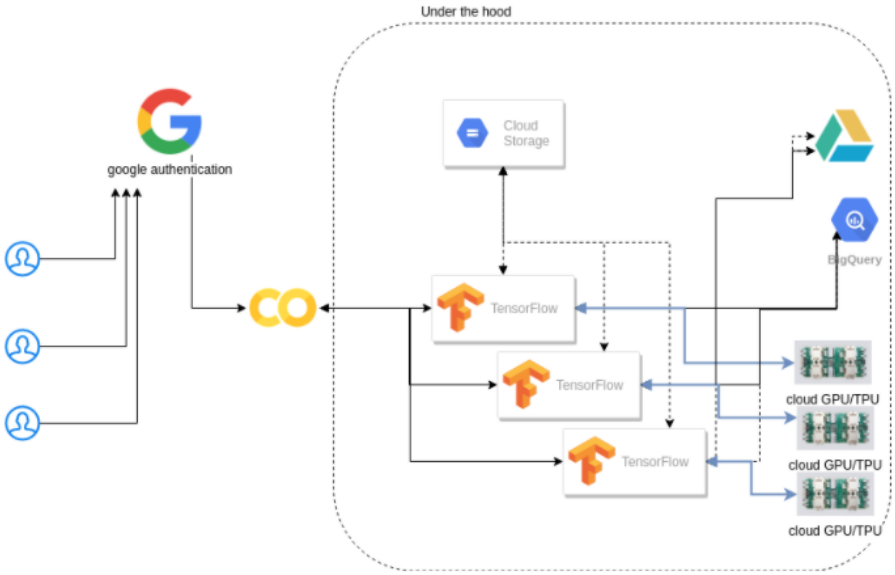

Figura 4.4 Schema architettura colab

4.3.1 Python

Per l'intero progetto è stato scelto come linguaggio di programmazione Python, in particolare Python3. (Melotti, 2016)Esso è un linguaggio di programmazione **Object-oriented** avente notevoli vantaggi: innanzitutto il vantaggio maggiore è legato alla facilità con cui è possibile reperire molte librerie in giro per la rete, soprattutto nell'ambito del machine learning, ed inoltre ha il vantaggio di essere estremamente portabile. Dal momento che si tratta di un **linguaggio interpretato**, è possibile usarlo su qualsiasi piattaforma su cui è installato l'interprete Python.

In Python, a differenza di quanto avviene in altri linguaggi analoghi, la memoria non viene gestita dal programmatore, ma verrà gestita automaticamente da un Garbage collector che si occuperà di allocarla e rilasciarla esonerando il programmatore dal seguente compito.

Per poter usare Python si deve prima installare l'ambiente di sviluppo sul dispositivo, oppure è possibile evitare tale step nel caso in cui si decida di utilizzare Google colab.

Un file scritto in Python sarà caratterizzato dall'estensione .py, e potrà essere eseguito all'interno dell'interprete Python digitando il comando

```
python script.py
```

A differenza di altri linguaggi di programmazione che delimitano blocchi di codice con parentesi graffe, in Python si usa il sistema delle indentazioni per delimitare blocchi di codice, i quali inizieranno tramite il carattere ":". Di seguito un esempio:

```
            for texts in data_json:
                    summary = str(texts['text']).strip()
                    summary = re.sub(r"\s", " ", summary)
                    data += summary + "  "
                f.write(data)
```

Tabella 4.2 Esempio di ciclo for in Python. Si nota la struttura senza parentesi graffe.

I tipi di dato non devono essere specificati direttamente, ma vengono determinati in maniera automatica dall'interprete. I tipi che python possiede sono i classici tipi, ed inoltre è possibile utilizzare una variabile senza che vi sia la necessità di dichiararla prima. Nell'esempio sopra riportato la variabile summary, infatti, viene usata anche se non è stata ancora definita esplicitamente.

Come detto in precedenza, uno dei vantaggi legati all'uso di Python riguarda la facilità con cui è possibile reperire sia librerie sia veri e propri framework da poter utilizzare per un determinato scopo. Nel momento in cui si decide di voler utilizzare una determinata libreria, essa deve essere installata sul dispositivo. **Pip** è un tool che viene usato per l'installazione, aggiornamento oppure disinstallazione di un determinato package.

Nel momento in cui viene eseguito, come si evince anche da Figura 4.5, verranno automaticamente scaricate dalla rete sia le varie librerie sia tutte le dipendenze che esse richiedono per poter essere utilizzate. Nell'esempio si è scelto di voler usare una versione specifica della libreria, ma qualora non fosse specificato nessun parametro allora l'installer provvederà a procurarsi la versione più aggiornata di tale libreria.

```
!pip install transformers==4.2.2

Collecting transformers==4.2.2
  Downloading transformers-4.2.2-py3-none-any.whl (1.8 MB)
     |████████████████████████████████| 1.8 MB 5.5 MB/s
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/di
Collecting tokenizers==0.9.4
  Downloading tokenizers-0.9.4-cp37-cp37m-manylinux2010_x86_64.whl (2.9 MB)
     |████████████████████████████████| 2.9 MB 44.9 MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
Collecting sacremoses
  Downloading sacremoses-0.0.46-py3-none-any.whl (895 kB)
     |████████████████████████████████| 895 kB 57.3 MB/s
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/d
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/pytho
```

Figura 4.5 Esempio di una schermata di pip install

Inoltre, dopo aver installato un package, per poterlo utilizzare all'interno dello script esso deve essere importato utilizzando il comando `import nome_modulo`

Nel seguente caso, l'intero modulo verrà importato e quindi utilizzabile. In altri casi si vorrebbe voler caricare in memoria soltanto determinati componenti, e non l'intera libreria. È possibile ottenere tale risultato nel seguente modo:

```
from nome_modulo import componente1,…,componenteN
```

Tabella 4.3 Comando per effettuare import di componenti in Python

In questo modo verranno effettivamente importati soltanto gli specifici componenti di cui avremo bisogno.

4.3.2 Librerie utilizzate

Nel corso del progetto ci si è affidati a diverse librerie per aiutarci nella fase di sviluppo, con il vantaggio di focalizzarci maggiormente sull'algoritmo ad alto livello, affidando i vari dettagli implementativi a librerie di terze parti.

Per l'implementazione della rete, in particolare ci si è affidati ad ***Huggingface***, una libreria che cerca di implementare tutti i vari modelli di reti neurali basate su transformers ed offrendo dei modelli pre-addestrati utilizzabili direttamente da parte degli utenti. Online si può trovare sia la repository ufficiale sia la documentazione ufficiale in cui viene spiegato sia come poter installare tale libreria sia come usarla nel modo migliore.

Ha il vantaggio di essere molto flessibile dal momento che risulta essere compatibile sia con PyTorch sia TensorFlow. Il comando da eseguire per installare la libreria è pip install transformers, e di default verrà scelto TensorFlow. Vengono messe a disposizione diverse classi standard per ogni modello, che sono Tokenizer, modello e configurazione per la rete. È possibile utilizzare dei modelli preaddestrati usando il comando from_pretrained() che si occuperà di scaricare e di caricare all'interno dei vari oggetti i vari hyper-parameters, il vocabolario del tokenizer ed infine i pesi di partenza della rete.

Capitolo 5

Architettura

Da un punto di vista schematico l'applicativo presenta una architettura molto lineare, ed i dati si muovono attraverso di essa tramite una pipeline. Vediamo ora come sono costituiti i vari blocchi dell'applicazione e come avviene lo scambio di dati tra i vari moduli.

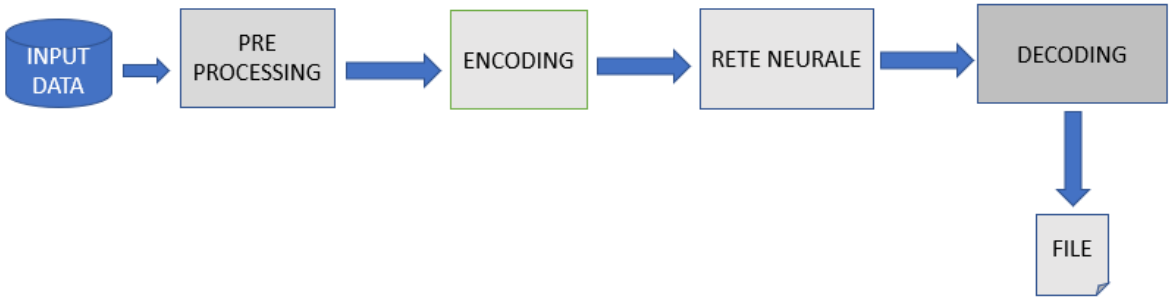

Figura 5.1 Schema riassuntivo architettura di rete

Il modulo principale della seguente applicazione è costituito dalla rete neurale. Una rete neurale, così come ho detto in precedenza è costituita a sua volta da vari layer che la rendono in grado di convertire un generico input in una rappresentazione tensoriale, quindi di tipo matematico. La parte fondamentale del processo decisionale è quindi caratterizzata dalla scelta dell'architettura di rete da voler utilizzare.

Lo stato dell'arte definisce infatti vari modelli, che come si è visto in precedenza ognuno è caratterizzato dai propri vantaggi e svantaggi. C'è quindi il bisogno di riuscire a trovare un compromesso tra prestazioni e risorse computazionali che si ha a disposizione.

Dopo una fase di studio si è deciso di affidarsi alla rete GPT-2. Essa presenta una architettura basata su transformers, e risulta essere molto usata per task legati al NLP con score molto elevati. In rete è possibile trovare dei modelli pre-addestrati che è possibile utilizzare come punto di partenza per il training, ed effettuare semplicemente una operazione di fine-tuning, cercando di lavorare o soltanto sui livelli più specifici della rete, oppure indipendentemente su tutti.

Possiamo distinguere due diverse pipeline, caratterizzate da un diverso flusso operazionale: il primo è legato al training della rete, e l'altro al suo effettivo uso per la generazione di contenuto, come in Figura 5.1.

5.1 Modulo Training

5.1.1Input

Per poter effettuare il training della rete abbiamo bisogno innanzitutto di un dataset da cui partire. Come ho già detto in precedenza, più esso è vasto e maggiore sarà l'accuratezza con cui si riuscirà a generare dati.

Il dataset nel nostro caso è costituito da testo prelevato da internet, e soprattutto esso non è etichettato.

Come ho spiegato in precedenza, i dati vengono ottenuti utilizzando scrapy, e sono organizzati in formato JSON, in cui ad ogni recensione viene assegnato un oggetto JSON.

5.1.2 Preprocessing

Prima di essere realmente utilizzati, tali dati devono essere sistemati, si deve rimuovere eventuali caratteri speciali, ed eventuali altri caratteri che potrebbero alterare il processo decisionale della rete. Nel nostro caso, il testo ottenuto come output da scrapy era composto da una lista di stringhe come si vede in Figura 5.2, dal momento che veniva generato del testo per ogni paragrafo. Il primo step, quindi, è stato quello di concatenare tutte queste stringhe in un'unica, e di produrre in output un documento JSON che sia codificato in UTF-8. Infine, prima di utilizzare tale file JSON in input si è provveduto ad epurare il file di caratteri di tabulazione superflui.

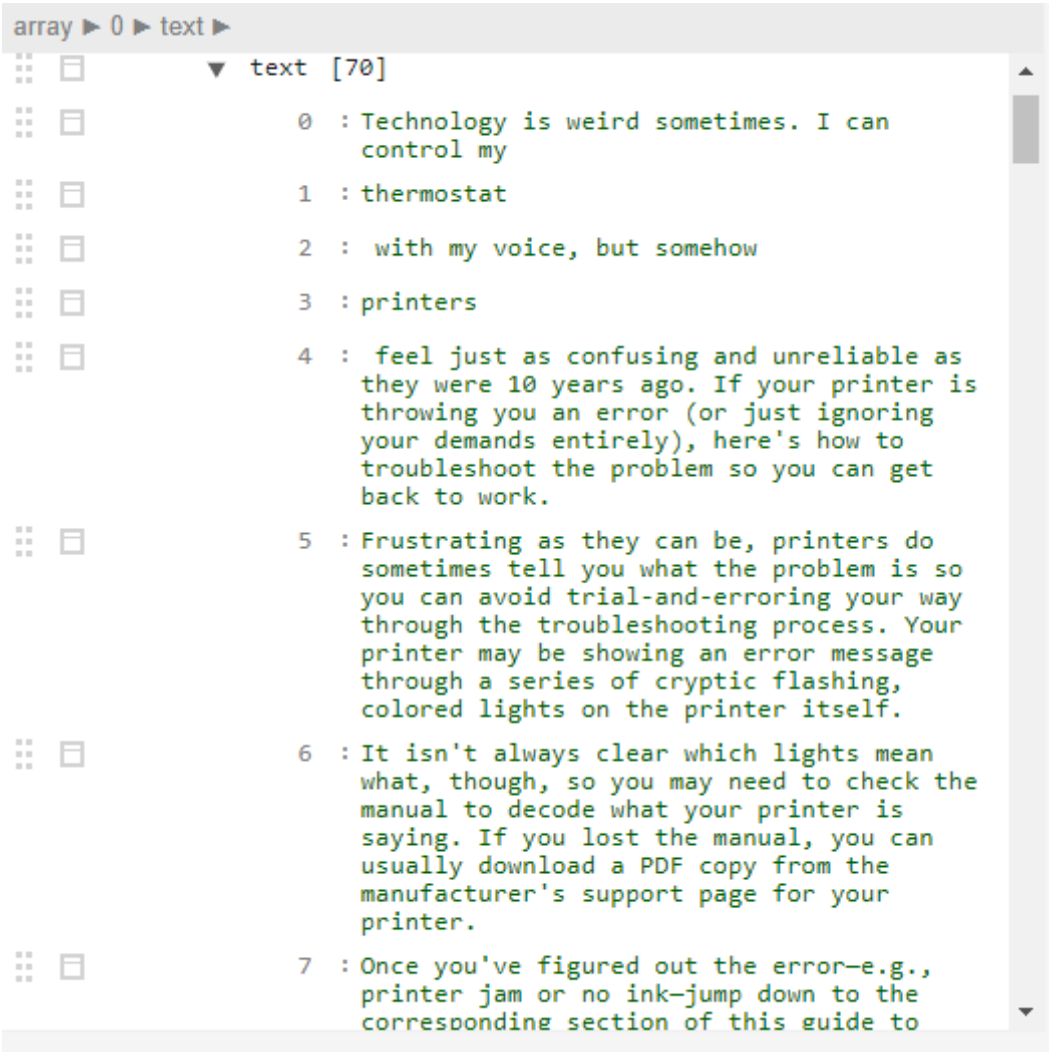

Figura 5.2 Esempio di documento JSON dopo esser stato estratto usando SCRAPY. In dettaglio la lista di stringhe di cui si compone ogni documento

5.2Architettura della Rete

Il modulo principale dell'applicativo è costituito dalla rete neurale. Come tutte le reti neurali ha due principi di funzionamento, uno in cui viene effettivamente addestrata ed un altro in cui riesce a produrre l'output desiderato a partire da un determinato input.

Il primo step che deve essere effettuato quando si vuole implementare delle applicazioni che sfruttino reti neurali è quello di decidere quale sia l'architettura della rete, la quale può essere sia una architettura già esistente, oppure un'evoluzione oppure una rete con architettura totalmente nuova. Probabilmente il collo da bottiglia è costituito principalmente dalle limitazioni imposte dai dispositivi fisici che si hanno a disposizione. Infatti, più l'hardware è potente, e più si può spingere sulle reti per poter usare architetture sempre più performanti.

Nei capitoli precedenti mi sono soffermato sulle diverse architetture di reti che vengono usate nell'ambito della generazione di testo automatica, spiegando sia il principio di funzionamento che c'è dietro ognuno di essi sia vantaggi e svantaggi. Bisogna fare una scelta su quale architettura utilizzare, e si è deciso di optare per GPT-2, seppur consci del fatto che probabilmente utilizzare la nuova, e più performante, GPT-3 avrebbe offerto prestazioni superiori. Si è deciso di usare GPT-2 invece di una classica rete ricorrente sia per via delle prestazioni più elevate nella generazione di testo, sia per la maggior semplicità nella fase di training della rete.

5.2.1 Training Dataset

La rete GPT-2 è nata con lo scopo di riuscire a predire la parola successiva all'interno di una frase data la stringa corrente, ragion per cui essa può essere usata senza problemi anche per poter generare del testo coerente. Per il training della rete originale è stato usato il dataset (A. Radford, 2019) **WebText**, un dataset costituito da un enorme quantità di documenti estrapolati dalla rete, circa 8M ed approssimativamente 40GB di testo.

Nella Figura 5.3 viene mostrato come variano le prestazioni del modello al variare del training dataset e della rete stessa. Nella tabella sono presenti le quattro versioni di GPT-2 e per ognuna di esse viene visualizzato lo score ottenuto da tale rete per ognuno dei dataset, e soprattutto senza che venga effettuato fine-tuning.

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPB)	text8 (BPC)	WikiText103 (PPL)	1BW (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	**21.8**
117M	**35.13**	45.99	**87.65**	**83.4**	**29.41**	65.85	1.16	1.17	37.50	75.20
345M	**15.60**	55.48	**92.35**	**87.1**	**22.76**	47.33	1.01	**1.06**	26.37	55.72
762M	**10.87**	**60.12**	**93.45**	**88.0**	**19.93**	**40.31**	**0.97**	**1.02**	22.05	44.575
1542M	**8.63**	**63.24**	**93.30**	**89.05**	**18.34**	**35.76**	**0.93**	**0.98**	**17.48**	42.16

Figura 5.3 Risultati estratti da Language Models are Unsupervised Multitask Learners, dove si mostra come variano le prestazioni utilizzando diverse versioni di GPT-2 ed un diverso dataset

5.2.2 Encoding

Per la codifica delle informazioni, in GPT-2 si utilizza **BPE** (Byte Pair Encoding), che è un algoritmo di codifica e di compressione attraverso cui si va a rimpiazzare tutte quelle sequenze di byte che hanno molte occorrenze all'interno dei dati con un altro simbolo.

All'interno del vocabolario vi saranno, quindi, soltanto quelle parole che si presentano maggiormente nel testo, mentre parole non molto usate verranno scomposte in diversi token. Cosa è un token? Un token può esser visto come l'unità elementare di una frase. Il modo più semplice per fare ciò è attraverso i blanc space quindi ogni token sarà una parola delimitata a destra e sinistra da uno spazio.

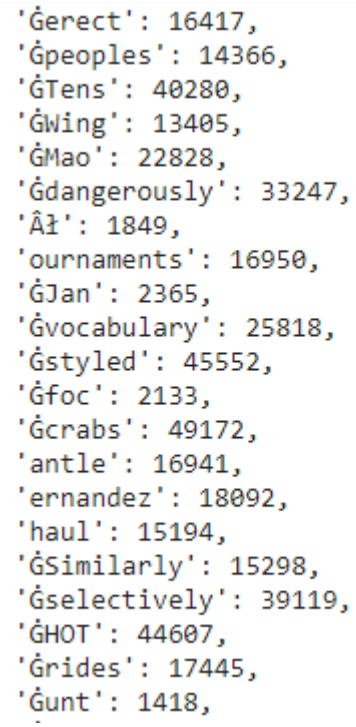

Figura 5.4 Esempio di vocabolario GPT-2, sulla destra vi è la frequenza della parola ed a sinistra il token

Tale lista nella Figura 5.4 viene utilizzata dal tokenizer per scomporre le parole in input in token di lunghezza variabile; quindi, ci saranno token di lunghezza diversa. Il vantaggio della scomposizione in token è che è possibile avere un meccanismo rapido che ci permetta da passare da un dominio ad un altro, riducendo di conseguenza la quantità di testo con cui dobbiamo aver a che fare. Inoltre, sarà sempre possibile effettuare l'operazione inversa grazie al vocabolario, la cui dimensione del vocabolario in GPT-2 è fissata a 50257 parole.

5.2.3 Fine Tuning

Un ruolo molto importante è dato alla fase di training. Quindi la definizione di un corpus da dare in pasto alla rete per provocare un aggiornamento dei vari parametri della rete fino al completamento della fase di Train.

In base a come viene addestrata la rete si riesce ad ottenere una sua diversa specializzazione nella qualità del testo generato. Sono stati definiti due diversi casi di funzionamento: nel primo caso si è deciso di adottare un'architettura a singola rete che sia in grado di generare testo per entrambe le classi, mentre nel secondo caso si è optato per una rete diversa per ogni categoria merceologica.

I vantaggi legati al fine-tuning sono enormi. Innanzitutto, si riduce il tempo necessario per effettuare il training di una rete neurale che in condizioni normali è un processo molto lungo e tedioso, dal momento che si deve far in modo di rendere la rete quanto più affidabile possibile nella generazione di testo.

Per questo motivo, è buona prassi effettuare il processo di generazione di testo partendo da un modello pre-trained. Il modello di partenza è la versione base di GPT-2. È possibile reperire tali modelli su Kaggle oppure eventualmente come moduli all'interno di librerie create ad hoc, come nel caso del package Python huggingface in cui sono rese disponibili al suo interno delle implementazioni dei modelli basati su transformers più diffusi e più importanti.

Come è stato spiegato in precedenza, esistono diverse versioni del medesimo modello di rete GPT-2 in taglie small, medium, large ed extra-large. Come si può evincere dal nome, esse sono basate su una medesima architettura, ma con un differente numero di layers, e quindi di parametri, che si traduce in prestazioni diverse.

Nel nostro caso si è deciso di adottare la versione medium, che risulta essere un buon compromesso in termini di prestazioni/memoria, e soprattutto per via delle limitazioni hardware imposte dall'environment utilizzato.

5.2.4 GPT-3

Le cose cambiano se si decide di usare GPT-3: dal momento che OpenAI ha reso disponibile la rete soltanto attraverso opportune API, se la si vuole utilizzare si deve necessariamente utilizzare un modello di comunicazione di tipo client-server, in cui il client si occuperà di fare richieste al server, il quale risponderà nel modo opportuno nel momento in cui avrà terminato il task.

Bisogna quindi tenere a mente tutti i problemi legati alla programmazione distribuita, soprattutto riguardo come vengano serializzati i dati all'interno dei pacchetti, e quale sia il formato richiesto dal server. Utilizzando il formato JSON non si corre nessun problema.

Con GPT-3, quindi, non ci troveremo a lavorare direttamente con la rete, ma ci sarà l'API che si interporrà tra un generico host e la rete. La rete, inoltre, non è upgradabile, ma utilizzabile nella versione base pre-addestrata. Un vantaggio legato all'utilizzo di quest'ultima rispetto a GPT-2 è che ci si disinteressa da tutti i problemi legati alla

gestione della rete, i quali saranno demandati ad OpenAI che li gestirà al posto nostro generandoci l'output appropriato con il minimo sforzo. Inoltre, non ci si deve dimenticare che se si decide di usare quest'ultimo approccio i requirement a livello hardware del dispositivo possono essere anche minimi, dal momento che non sarà più necessario dover effettuare calcoli complessi che saranno tutti svolti sui server di OpenAI.

5.3 Memorizzazione

Subito dopo aver generato del testo, si deve decidere in che formato memorizzare i dati. In linea di principio qualsiasi standard andrebbe bene per la memorizzazione dei dati, da un semplice formato testuale non formattato fino ad arrivare a formati molto più complessi come il JSON.

Dopo attente analisi si è deciso di usare quest'ultimo, soprattutto dal momento che esso è diventato lo standard più usato nell'ambito dei data exchange e del networking. Quindi, salvando già i dati in quest'ultimo formato li rende più facilmente accessibili per futuri step. Inoltre, un altro vantaggio legato a quest'ultimo formato è anche la presenza di librerie open source che consentono di serializzare e deserializzare il contenuto del file all'interno di variabili Python, non dovendo quindi gestire in maniera manuale il processo di lettura/scrittura su file.

Quest'ultimo step non richiede molte risorse computazionali, a differenza di quanto si verificava nel caso della rete, che richiedeva molte risorse per poter essere eseguita. Pertanto, si riesce ad eseguire il modulo di memorizzazione anche in locale, non necessariamente su server dal momento che si tratta di un task di breve durata e che verrà eseguito una tantum.

Capitolo 6

Pipeline

Dopo aver parlato nei capitoli precedenti di nozioni teoriche, dell'architettura del modello utilizzata e di tutte le tecnologie utilizzate all'interno del progetto, è ora di focalizzarsi su aspetti più pratici ed implementativi.

In particolare, nel seguente capitolo verranno trattati aspetti pratici riguardo l'implementazione del generatore di testo, ed inoltre, per ogni difficoltà trovata si cercherà di spiegare come essa sia stata superata. Tutti i vari step della pipeline verranno trattati separatamente in ogni paragrafo, fornendo se possibile oltre ad una spiegazione anche frammenti di codice sorgente, per rendere il tutto più comprensibile.

6.1 Raccolta di Dati

Il primo step che è stato fatto è stato quello di raccogliere i dati dalla rete, in particolare si è prediletto dati provenienti da siti web che facessero recensioni di dispositivi tecnologici, nel nostro caso ne abbiamo scelto solo uno, pcmag.com, un sito molto fornito, e soprattutto con delle recensioni molto accurate.

A questo punto si è optato per la scrittura di un web crawler che in maniera automatica navigasse attraverso i vari collegamenti ipertestuali e salvasse il contenuto di nostro interesse.

Uno strumento molto utile durante questa fase è stato quello di ispezionare il codice sorgente della pagina web in modo da poter capire come fare per poter navigare attraverso i vari collegamenti, e soprattutto come estrarre il testo dagli articoli.

Per la navigazione attraverso le varie pagine si sono dovuti seguire due step:

- In Figura 6.1 viene evidenziato il primo, in cui per poter aprire la pagina successiva non si deve far altro che seguire il link contenuto all'interno del titolo; quindi, è stato sufficiente salvarsi da qualche parte la classe CSS dell'elemento specifico.
- Il secondo step, invece, consiste nell'aprire la pagina successiva dove sono presenti tutti i riferimenti alle varie recensioni. Tale pagina la si può trovare all'interno di una lista di riferimenti (dettaglio in Figura 6.2); in particolare è stato sufficiente cliccare sulla freccia "Next" fintanto che è presente un'altra URL.

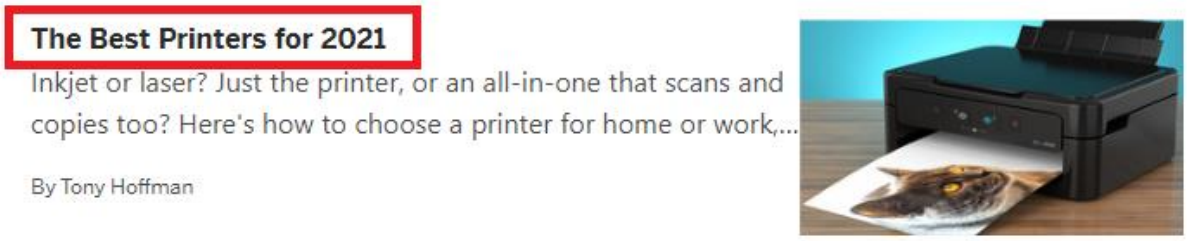

Figura 6.1 Dettaglio prelevato dalla pagina web

Figura 6.2 Lista riferimenti

Da un punto di vista implementativo, il tutto è stato formalizzato nel seguente modo:

```
def parse(self, response):
        descr_page_link = response.xpath('//h2[@class="text-
base md:text-xl font-brand font-bold"]/a')
        yield from response.follow_all(descr_page_link,self.parse_item
)

        next_page_url = response.xpath('//li[@class="page-
item"]//a[contains(@rel,"next")]/@href').extract_first()
        print("{}:".format(next_page_url))
        if next_page_url is not None:
            yield scrapy.Request(response.urljoin(next_page_url))
```

Tabella 6.1 Funzione Parse del modulo web crawler

Nella prima parte del codice si segue tutti i link che rispettano il pattern passato in input, mentre nella parte seguente si clicca il tasto next fino a quando l'URL non è null.

L'obiettivo di un crawler non è solo quello di navigare tra le pagine, ma di effettuare qualche operazione, come nel nostro caso di salvare una copia dei dati presenti nelle pagine che visita. Per fare ciò è stato implementato un altro metodo, eseguito come callback sulle pagine che vengono aperte dal metodo parse, e che viene eseguito in maniera asincrona.

Una generica pagina di recensione è organizzata nel seguente modo: innanzitutto il contenuto viene delimitato dai tag HTML <article></article>. Tuttavia, non è sufficiente utilizzare tale tag per il filtraggio, ma se possibile si cerca di essere quanto più specifici possibile, in modo da prelevare soltanto ciò di cui abbiamo bisogno.

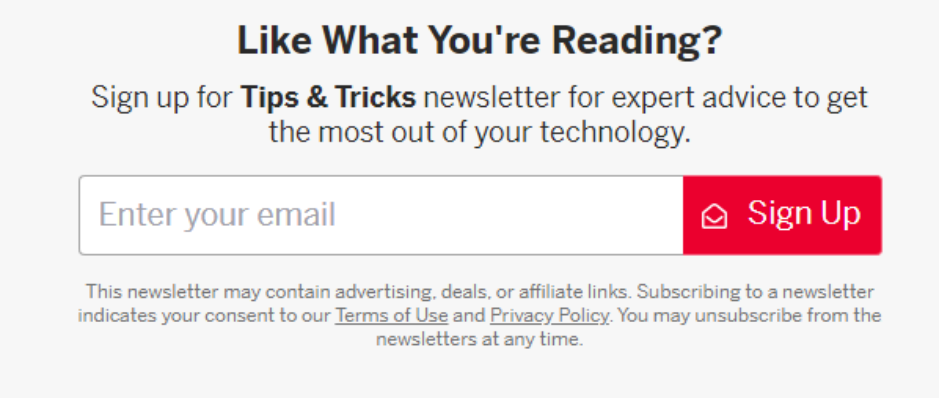

Figura 6.3 Descrizione in forma visiva di un problema avuto

Infatti, come si evince dalla Figura 6.3, un problema che abbiamo avuto è stato proprio legato al seguente popup all'interno dell'articolo, ragion per cui in ogni documento compariva in forma testuale la seguente dicitura.

Per risolvere tale problema si è cercati di essere quanto più restrittivi nel filtraggio del testo all'interno del documento, in modo da evitare contenuti non voluti.

```
def parse_item(self,response):
        for quote in response.xpath('//article[@class="w-full rich-
text"]'):
            yield{
                'text':quote.xpath('./p//text()').getall()
        }
```

Tabella 6.2 Implementazione del metodo parse_item() che si occupa di estrarre il testo dalle varie pagine web e salvarlo in formato testuale su file.

All'interno del seguente metodo si discende attraverso il DOM, partendo dal TAG article, e si va a convertire in testo tutto ciò che è contenuto all'interno di un paragrafo.

A questo punto, alla fine dell'esecuzione si otterrà in output un documento JSON organizzato come in Figura 5.2, e che, come ho già detto, deve essere sistemato prima di poterlo utilizzare.

Quindi, lo step successivo è stato quello di generare un file JSON in cui il valore dell'oggetto sia una stringa anziché una lista di stringhe. Per fare ciò si è usato il seguente script.

```
import json

filename= "new_printers.json"
f = open(filename)
data = json.load(f)
list = []
for index in range(0,len(data)):
```

```
    for key,value in data[index].items():
      list=" ".join(value)
      data[index] = {'text':list}


with open('data_new_printers_scrap.json', 'w', encoding='utf-8') as f:
    json.dump(data, f, ensure_ascii=False, indent=4)
```

Tabella 6.3 Frammento di codice per pre-processing del testo di input

tramite la libreria json si riesce a gestire in modo molto più semplice un file JSON, che viene caricato all'interno di una variabile con l'istruzione json.load(), e si riesce ad iterare su di esso utilizzando la notazione vettoriale, e ad iterare sulle liste presenti all'interno dei vari oggetti, e rimuovere tutti gli spazi vuoti.

Infine, verrà prodotto come output un nuovo file JSON avente un determinato nome, che verrà successivamente usato come corpus per la nostra rete.

6.2Rete Neurale

Come ho già detto in precedenza, la rete adottata è stata GPT-2. In particolare, si è deciso in un primo momento di utilizzare una sola rete neurale che fosse in grado di gestire entrambe le categorie merceologiche, e successivamente, si è deciso di effettuare uno split, e quindi avere una rete per ogni categoria.

Il problema maggiore legato alla seguente decisione è legato al dover creare un nuovo dataset di partenza. Infatti, il dataset originale era costituito dall'unione di tutti i documenti delle categorie "printer" e "smart tv". Quindi, oltre al seguente dataset sono stati definiti un dataset per le stampanti ed un altro per tutte le recensioni delle televisioni.

Nel nostro caso, come framework si è deciso di usare (huggingface, 2021) huggingface, una libreria python che offre una implementazione e dei modelli pre-addestrati per la maggior parte delle reti neurali basate su transformers, ed inoltre, con il vantaggio di poter avere elevata astrazione.

Quindi, partendo da un modello pre-addestrato è stato fatto un lavoro di fine tuning, riducendo in questo modo le dimensioni del dataset necessario per l'addestramento della rete.

6.2.1 Split Dataset

Il dataset iniziale, composto da dati non etichettati, deve essere diviso in due insiemi: uno per il train ed un altro per il testing. Lo split è stato effettuato sfruttando

il metodo train_test_split nella libreria (scikit-learn.org, 2021)sklearn. Si è deciso di utilizzare per il test set il 15% dei dati, selezionati in maniera random e senza rimessa.

Un problema su cui ci siamo dovuti scontrare è stato la limitazione imposta da Google Colab, in particolare la massima durata di un task all'interno di una macchina virtuale, fissata nel nostro caso a 12 ore. Per poter rientrare in tale intervallo di tempo il dataset di partenza è stato suddiviso in vari pezzi. In ogni iterazione si partiva dal checkpoint precedente utilizzando un altro dataset.

```
import re
import json
from sklearn.model_selection import train_test_split

with open('splitB.json') as f:
    data = json.load(f)

def build_text_files(data_json, dest_path):
    f = open(dest_path, 'w')
    data = ''
    for texts in data_json:
        summary = str(texts['text']).strip()
        summary = re.sub(r"\s", " ", summary)
        data += summary + "  "
    f.write(data)

train, test = train_test_split(data,test_size=0.15)
build_text_files(train,'train_dataset.txt')
build_text_files(test,'test_dataset.txt')
```

Tabella 6.4 Generazione di Train e test file a partire dal dataset iniziale. Da notare l'utilizzo di train_test_split per la generazione dei due subset.

Nel seguente frammento di codice viene riportato il processo di generazione di file train e test, ottenuti utilizzando train_test_split a cui si passa come parametri i dati che si vuole splittare e la dimensione del test_split.

Quindi, il training set sarà usato per addestrare la rete, mentre con il test set si valuterà l'accuracy e le altre prestazioni della rete. I dati vengono scritti all'interno di un file temporaneo in formato testuale, concatenando tutte le stringhe presenti nel corpus così da poter essere usato nei prossimi passi, così come mostrato in Figura 6.4.

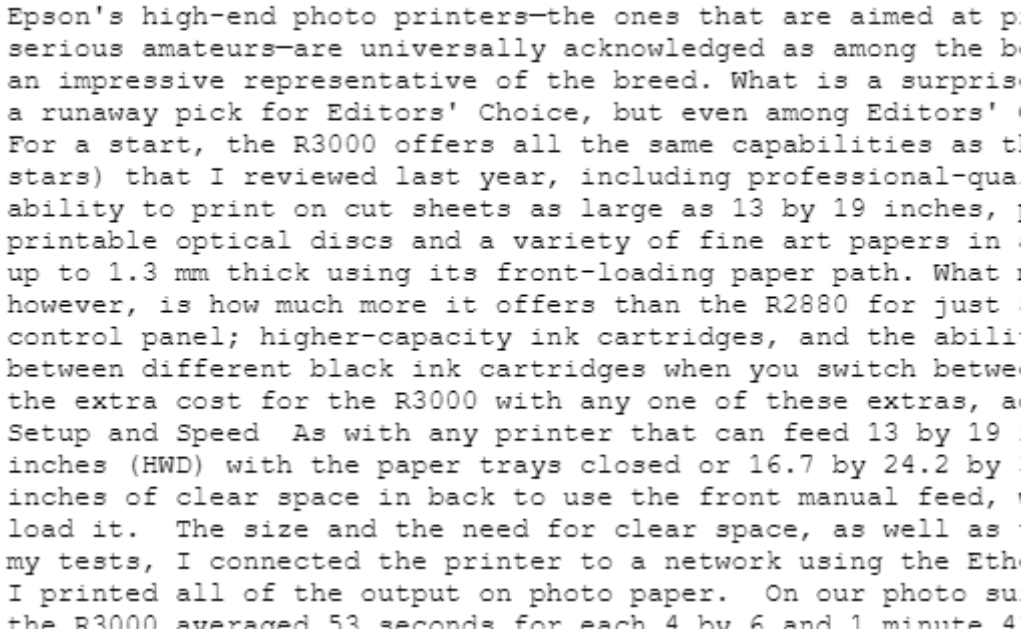

Figura 6.4 Esempio di file testuale del Training_set e test_set

Utilizzando il seguente stratagemma si è riusciti senza problemi ad addestrare la rete neurale utilizzando Colab, preoccupandosi soltanto di gestire manualmente il cambio di file in input del dataset e del modello da cui partire.

Il caricamento dei dati all'interno della rete viene gestito da un altro modulo, il DataLoader, il cui compito è quello di gestire e caricare in memoria in maniera sequenziale tutti i dati di cui abbiamo bisogno. I dati vengono suddivisi in blocchi dalla dimensione fissa, 128Byte nel nostro caso, ed ogni blocco di dati è trattato in modalità batch. Per poter costruire dei batch, viene fatto in maniera automatica un'operazione di padding (viene aggiunto un riempimento automatico fino a raggiungere la lunghezza del blocco) e di data augmentation.

La classe DataCollatorForLanguageModeling gestisce in maniera automatica il caricamento di dataset testuali in memoria, effettuando pad dinamico degli input di lunghezza inferiore a quella del block size.

In input riceve un tokenizer che verrà usato per codificare i dati in input ed il parametro mlm che nel nostro caso viene settato a false dal momento che esso viene utilizzato per task legati a masked language in cui in maniera randomica token in input verranno mascherati.

6.2.2 Tokenizer

Per il caricamento del tokenizer in memoria ci siamo affidati alla classe generica AutoTokenizer, la quale ha soltanto il metodo from_pretrained. Usando tale metodo verrà scaricato dalla rete e poi istanziato un oggetto di tipo tokenizer, selezionato in

base alla stringa passata come parametro al metodo. Essa identificherà il nome del modello, oppure il path del file di configurazione del tokenizer.

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")
```

Tabella 6.5 Frammento di codice con dettaglio di definizione di un tokenizer 'gpt2' utilizzando la classe AutoTokenizer.

Se provassi ad eseguire `tokenizer.get_vocab()` mi verrebbe restituito un oggetto di tipo dictionary che contiene il vocabolario del tokenizer di gpt2, così come viene mostrato anche in Figura 5.4.

6.2.3 Trainer

La libreria huggingface mette a disposizione un oggetto di tipo Trainer, che permette di gestire in maniera automatica ed indipendente dalla tipologia di rete, il processo di training. La classe è compatibile sia con Pytorch sia Tensorflow, anche se di default verrà sempre utilizzato Pytorch.
Internamente il trainer gestisce in maniera totalmente automatica il loop di training che normalmente si ha in una rete neurale, in cui si itera attraverso le varie epoche. Inoltre, viene gestito in maniera automatico l'aggiornamento delle variabili di loss, accuracy e tutti gli altri hyper parameters del network.

```
model = AutoModelWithLMHead.from_pretrained('./printers_out')
training_args = TrainingArguments(
    output_dir="./printers_out2", #The output directory
    overwrite_output_dir=True, #overwrite the content of the output di
rectory
    num_train_epochs=3, # number of training epochs
    per_device_train_batch_size=8, # batch size for training
    per_device_eval_batch_size=8,  # batch size for evaluation
    eval_steps = 200, # Number of update steps between two evaluations
.
    save_steps=400, # after # steps model is saved
    warmup_steps=250,# number of warmup steps for learning rate schedu
ler
    prediction_loss_only=True,
    )
trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=data_collator,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
)
```

Tabella 6.6 Frammento di codice con implementazione modulo Trainer.

Il trainer richiede come parametri il modello della rete, i vari parametri di configurazione provenienti dalla classe TrainingArguments, il riferimento al DataCollator che si occuperà di passare i dati alla rete in batch di dimensione fissata, ed infine si deve passare il training set ed il test set.

La classe TrainingArguments permette di configurare tutti quei parametri necessari per la corretta esecuzione di un loop di training. Il numero di epoche su cui è stato fatto il training è di 3 epoche, non si è potuto incrementare tale valore per una questione di tempo necessario per l'esecuzione del task.

Di default viene usato come optimizer (Diederik P. Kingma, 2014)Adam, il learning rate è fissato a 5e-5. Adam è un metodo che fa un Gradient descent.

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Figura 6.5 Adam: A method for stochastic optimization

Come si evince dalla Figura 6.5 verrà calcolata la derivata di x, l'hessiano, fino a quando non si raggiunge un punto di minimo che implicherà che l'errore che si commetterà nella predizione sarà sufficientemente basso.

Il momento del k-esimo ordine corrisponde al valore atteso di una generica variabile definito tramite l'equazione:

$$mk = \sum_{i=1}^{N} (x_i - m)^k p_i$$

Utilizzando il metodo trainer.train() inizia la fase di training della rete, che come si può immaginare è un processo molto lungo. Nel nostro caso ogni 400 step verrà creato un nuovo modello.

Step	Training Loss
500	2.665200
1000	2.467500
1500	2.335900

Nella seguente tabella sono stati riportati i valori di training loss ottenuti in seguito all'operazione di training della rete. La cosa che si nota è che il loss scende in maniera

molto dolce tra una iterazione ed un'altra. Infine, il modello verrà salvato usando l'istruzione trainer.save_model().

Per riuscire a trarre vantaggio dall'uso di Colab abbiamo integrato Google Drive all'interno della macchina virtuale, in modo che tutti i vari modelli fossero salvati automaticamente su cloud, e soprattutto direttamente accessibili e condivisibili in remoto.

Nel momento in cui un modello viene salvato sul device oltre ai vari risultati intermedi presenti nei vari checkpoint vi sarà presente il file config.json che è il file più importante dal momento che contiene la configurazione iniziale della rete e dei suoi vari parametri come si vede in Figura 6.6.


```
{
  "_name_or_path": "./printers_out",
  "activation_function": "gelu_new",
  "architectures": [
    "GPT2LMHeadModel"
  ],
  "attn_pdrop": 0.1,
  "bos_token_id": 50256,
  "embd_pdrop": 0.1,
  "eos_token_id": 50256,
  "gradient_checkpointing": false,
  "initializer_range": 0.02,
  "layer_norm_epsilon": 1e-05,
  "model_type": "gpt2",
  "n_ctx": 1024,
  "n_embd": 768,
  "n_head": 12,
  "n_inner": null,
  "n_layer": 12,
  "n_positions": 1024,
  "resid_pdrop": 0.1,
  "summary_activation": null,
  "summary_first_dropout": 0.1,
  "summary_proj_to_labels": true,
  "summary_type": "cls_index",
  "summary_use_proj": true,
  "task_specific_params": {
    "text-generation": {
      "do_sample": true,
      "max_length": 50
    }
```


Figura 6.6 Esempio di file config.json

Utilizzando il seguente file il Trainer è in grado di ripristinare uno stato precedente della rete. Il metodo from_pretrained(), infatti, può ricevere come parametro o una stringa che si riferisce ad un modello standard oppure il percorso della directory che contiene il file di configurazione che si vuole usare.

6.3Generazione di testo

Per poter generare del testo tramite GPT-2 la rete necessita di ricevere come input del testo da cui poter partire. Infatti, la rete inizialmente è stata creata per poter prevedere la prossima parola dato un generico input.

Nel nostro caso vi era il problema che tutti i vari dati in input erano organizzati in formato JSON, quindi come coppie di chiave valore.


```
"specs": {
  "print_speed_colour": "5.5ppm",
  "maximum_resolution": "1200 x 1200 DPI",
  "wi-fi": "Yes",
  "approximate_page_yield_a4,_colour": "120
    Pages",
  "scanner_features": "Front-Panel Scan",
  "duplex_print_options": "Manual",
  "printer_features": "Contains More Than 20%
    Recycled Plastic",
  "box_contents": "HP DeskJet 2724 All-in-One
    Wireless Inkjet Printer - HP 305 Black Ink
    Set-Up Cartridge - HP 305 Tri-Colour Ink
    Set-Up Cartridge - Power Cable - Set-Up
    Instructions",
  "wireless_technology": "Yes",
  "type": "Home Printer",
  "print_technology": "Inkjet",
  "dimensions_wxdxh": "154 x 425 x 191 mm (H x
    W x D)mm",
```


Figura 6.7 Esempio di dati di input in formato JSON

Si evince subito dalla Figura 6.7 come sono gestiti i dati. Innanzitutto, la chiave ci fornisce informazioni su qual è la feature, e quindi per poter generare testo si è cercato di concatenare chiave e valore. All'interno della chiave per poter delimitare diverse parole viene utilizzato come separatore il carattere "_", il quale dovrà essere convertito in un carattere di spazio vuoto. Il valore, invece, può essere una stringa oppure un booleano.

Un altro problema è legato al numero delle features presenti all'interno di un documento. Tipicamente non siamo interessati a tutte le features presenti, e magari ne vorremo soltanto un suo sottoinsieme. Inoltre, non tutti i documenti JSON avranno gli stessi fields. Infatti, è possibile che vi possano essere campi mancanti. Quindi, deve essere gestito anche il seguente caso all'interno della generazione di testo.

Riguardo il filtraggio, nel nostro caso è stato fatto staticamente selezionando le features che erano più rilevanti per ogni documento.

Il modulo che si occupa di generare testo non farà altro che iterare fino a quando non saranno terminati tutti gli elementi in input. Innanzitutto, si deve definire il token iniziale da cui cominciare con la generazione del testo.

6.3.1 Rete GPT-2

La rete addestrata in precedenza si occuperà di generare il testo sulla base dell'input ricevuto. Si è deciso di usare l'oggetto pipeline definito nella libreria transformers che permette di gestire in maniera astratta un gran numero di task, il quale deve essere passato come parametro. Sulla base del task passato come input verrà istanziata una determinata classe, nel nostro caso ("`text-generation`") sarà restituito un oggetto di tipo TextGenerationPipeline. Tale pipeline si occuperà in automatico di codificare l'input, e di decodificare l'output generato dalla rete. Otterremo come valore di ritorno un oggetto di tipo dict al cui interno saranno presenti i seguenti campi:

- Generated_text, che contiene il testo generato dalla pipeline
- Generated_token_ids, che rappresenta i token id del testo generato

```
from transformers import pipeline
generator = pipeline('text-generation', model='./printers_out2')
```

Tabella 6.7 Dettaglio del codice utilizzato per effettuare la generazione del testo, usando l'oggetto pipeline.

I parametri che sono stati passati in input alla pipeline sono la tipologia di task ed il modello che si vuole utilizzare per la generazione del testo. Il nome del modello costituisce il path in cui è salvato in memoria il modello.

Dovrebbe essere passato come parametro anche il tokenizer, tuttavia nel caso in cui non si provveda a specificarne uno esplicitamente il sistema sarà in grado di procurarsi il tokenizer di default adatto per la tipologia di modello che si sta usando.

La variabile generator viene usata come oggetto funzionale, ed è quindi possibile invocare la pipeline per la generazione di testo semplicemente nel seguente modo:

```
temp = generator(sentence, max_length=50)[0]['generated_text']
```

Tabella 6.8 Generazione del testo e memorizzazione del risultato in una variabile temporanea.

passiamo come input alla rete la stringa e la lunghezza massima misurata in numero di token che si desidera avere in output. È possibile definire anche altri vincoli sul testo che si vuole generare, come ad esempio la lunghezza minima del contenuto che si vuole generare. Tramite il parametro "temperature", per esempio, possiamo andare ad

impostare un valore minore di 1 che sarà usato per la generazione del prossimo token, oppure possiamo usare il parametro "top_k" per definire un valore di soglia per selezionare i k token più probabili dato il token attuale.

In maniera indipendente verrà generato un pezzo di testo alla volta, e concatenati l'uno con l'altro. Per ogni generazione del testo verrà salvato soltanto quel testo che soddisfi la seguente condizione:

```
if temp.count(".")>1:
      last_dot = temp.rfind('.')
      text = temp[0:last_dot+1]
      print(text)
    else:
      text=""
```

Tabella 6.9 Modulo generazione del testo, con dettaglio sul conteggio di quante frasi complete sono state prodotte in output.

Non siamo quindi interessati al caso in cui si abbiano frasi non complete, ed inoltre, è stato deciso che il numero minimo di frasi deve essere fissato a due. Se tale vincolo è rispettato allora si procederà a salvare nella variabile text il testo prodotto in output dalla rete.

Come ho detto in precedenza, le features sono selezionate in maniera statica

```
for i,j in data[count]["specs"].items():
      if i in list:
           do_something...
```

Tabella 6.10 Un esempio di codice che può essere usato per iterare sulle specifiche di un documento JSON e selezionare solo quelle che hanno lo stesso nome di quelle nella lista.

Nel caso in cui la chiave i-esima sia contenuta nella lista allora verrà fatta partire la pipeline sul testo generato concatenando chiave e valore effettuando un opportuno preprocessing. Analogamente a quanto detto prima si verificherà la condizione precedente, e concateneremo la stringa attuale a quella precedente.

```
        tmp = str(i)
        z = tmp.replace('_',' ')
        #devo definire un token iniziale
        start = "The "
        sentence = start + z+" "
        connector = ["is of","is a","is an"]
        if re.match("[0-9]\w+",j):
          sentence = sentence + connector[0] +" "+ j
```

```
elif re.match("[aeiouh]\w+",j):
  sentence = sentence + connector[2] +" "+ j
elif j=="Yes":
  sentence="It has "+z
elif j=="No":
  sentence="It does not have "+z
else:
  sentence = sentence + connector[1] +" "+ j
```

Tabella 6.11 Generazione statica della frase iniziale da inviare alla rete.

Per il preprocessing si è cercato di sfruttare le espressioni regolari per determinare come unire le varie parole. Il metodo è molto semplice anche se produce output che non sono molto variabili e che presentano sempre la stessa struttura. Si è deciso, infatti, di fare iniziare un nuovo periodo nel seguente modo, e fissando la lunghezza massima del testo generato a 50 token si riesce a contenere la lunghezza complessiva dell'output.

Il processo di generazione ha una durata variabile, e dipende principalmente dal numero di documenti che si vuole generare. Al termine del processo di generazione del testo il tutto verrà scritto su file in formato JSON per una questione di comodità, come mostrato in Figura 6.8.

```
array [185]

  0 : The printer HP deskjet 2724 has a price of 49.98€ ($58.36 at
      Amazon) , so you have a few choices. The print speed colour
      is a 5.5ppm monochrome page and the average for an image is
      8.6ppm. The dimensions wxdxh is of 154 x 425 x 191 mm (H x W
      x D)mm and weighs 129 pounds. The frame is built around a 3.
      The standard interfaces is a Usb, Usb 2.2, and PostScript
      driver. Paper handling consists of an 80-sheet main input
      tray and a 100-sheet multipurpose tray. The weight is a
      3.4kg, which isn't quite as impressive as the  Dell B5460dn
        ($7,499.00 at Dell Technologies)  , our Editors' Choice
      monochrome laser AIO. The minimum system requirements is a
      Windows 7 & 10 - MacOS 10.12 (Sierra) or Later PC OS.

  1 : The printer HP officejet pro 9022 has a price of 249.76€ by
      17.78cents and weight of 3.75kg. The print speed colour is
      of 20ppm (ppm) for black and 10ppm for red. In our testing,
      the ET-2650 churned at a smooth 25.5ppm.

  2 : The printer HP deskjet 2710 has a price of 39.98€ for
      monochrome pages and 9.1 by 17.9 by 19.9 inches. The print
      speed colour is a 5.5ppm.  I tested it over Ethernet from
      our standard Intel Core i5 PC running  Windows 10
      Professional . The print technology is a Thermal inkjet. It
      uses ink crystals to form a print surface that is heated by
      liquid glycol, and to apply pressure that is dissolved in a
      chemical reaction with an ultraviolet laser. The standard
      input capacity is of 60 Sheets, expandable to 900 sheets.
```

Figura 6.8 Output generato dalla rete

6.3.2 Rete GPT-3

In un secondo momento si è deciso di sperimentare con (OpenAI, 2021)GPT-3 per la generazione del testo. Come ho già detto in precedenza, esso è un modello nuovo, rilasciato per la prima volta solo nel 2020, ed accessibile per ora soltanto in versione BETA, utilizzando opportune API per fare richieste e ricevere il risultato.

Tale rete, quindi, non può essere modificata né addestrata con un diverso dataset, ma ne è reso disponibile l'utilizzo solo sulla rete pre-addestrata, utilizzando un modello di tipo client-server.

Per poter comunicare con il server un client deve essere in possesso di una API key, univoca per ogni utente, la quale sarà inserita all'interno del pacchetto di REQUEST dal client stesso.

Il server genererà una risposta, in formato JSON che sarà inviata al Client nel momento in cui il task sarà terminato. Vediamo un esempio di risposta del server:


```
{
  "id": "cmpl-GERzeJQ4lvqPk8SkZu4XMIuR",
  "object": "text_completion",
  "created": 1586839808,
  "model": "davinci:2020-05-03",
  "choices": [{
    "text": " of reading speed. You",
    "index": 0,
    "logprobs": null,
    "finish_reason": "length"
  }]
}
```


all'interno del response possiamo notare il campo model che specifica quale modello è stato utilizzato per la generazione del testo. Vengono messi a disposizione dall'API diversi **engine** che permettono di accedere ad un diverso modello, ognuno dei quali con diverse caratteristiche. Di base vi saranno alcuni modelli molto generici ed altri molto specifici.

Tra i vari modelli non si può non citare davinci, curie, ada e babbage. Essi sono stati pensati per task legati al NLG, ed in particolare ognuno di essi risulta essere più appropriato per una determinata tipologia di task. Davinci è il modello più generico e con le performance medie migliori rispetto gli altri, che sono molto più performanti su task specifici.

Per effettuare la richiesta al server si userà il metodo

```
response = openai.Completion.create(
                    engine="davinci-instruct-beta",
                    prompt=toSumbmit,
                    temperature=0.7,
                    max_tokens=1000,
                    top_p=1.0,
                    frequency_penalty=0.0,
                    presence_penalty=0.0
                )
```

Tabella 6.12 Dettaglio sulla creazione di un oggetto di tipo Completion, con cui avviare una connessione HTTPS con il server.

Completion è un endpoint utilizzabile per la maggior parte dei task di generazione di testo. Il parametro prompt può essere una stringa oppure una lista di stringhe.

GPT-3 introduce un nuovo elemento, ovvero la capacità da parte del programmatore di poter passare in input alla rete anche una serie di istruzioni scritte in inglese che saranno interpretate dalla rete ed utilizzate per cercare di ottenere il miglior risultato possibile.

```
prompt = "Write a printer review based on these notes:\n\n"
```

Tabella 6.13 Dettaglio nel codice che evidenzia istruzioni scritte in inglese che possono essere interpretate dalla rete per generare un risultato basato sulle specifiche.

Nel nostro caso l'istruzione utilizzata è stata la seguente, seguita dalla lista di chiave-valore dei vari elementi dell'oggetto JSON, concatenati in una sola stringa ed inviati in un'unica HTTP GET request al server. Analogamente a quanto fatto con GPT-2, quindi, abbiamo generato i vari documenti e salvato il tutto in formato JSON.

L'unico problema avuto è stato legato al fatto che è stata utilizzata una versione trial di OpenAI gpt-3, ragion per cui era limitata la quantità di documenti generabili e soprattutto la durata dei task. Quest'ultimo problema è stato risolto dividendo in chunk l'input, e generando N file di output, ognuno per ogni chunk.

Infine, con una operazione di merge i vari files sono stati uniti in un singolo dizionario ed infine scritti su un nuovo file JSON. Per risolvere alcuni problemi legati ad errori legati alla formattazione JSON sono stati usati tool disponibili in rete, in particolare un tool disponibile sul sito web https://jsonformatter.curiousconcept.com/ che è stato usato sia per validare sia per correggere eventuali errori di formattazione.

6.3.3 Merge

Il modulo che si occupa di effettuare l'operazione di merge sfrutta la libreria glob, in particolare il metodo `files=glob.glob("*.json")` che mi ritorna la lista di tutti i files presenti all'interno di una directory che rispettano un determinato pattern, nel seguente caso tutti i file aventi estensione JSON. Partendo dalla lista dei files non si farà altro che aprire un file alla volta in lettura, leggerne il contenuto serializzato in formato JSON e salvarlo all'interno di una lista. Infine, lo step finale riguarderà la scrittura su file.

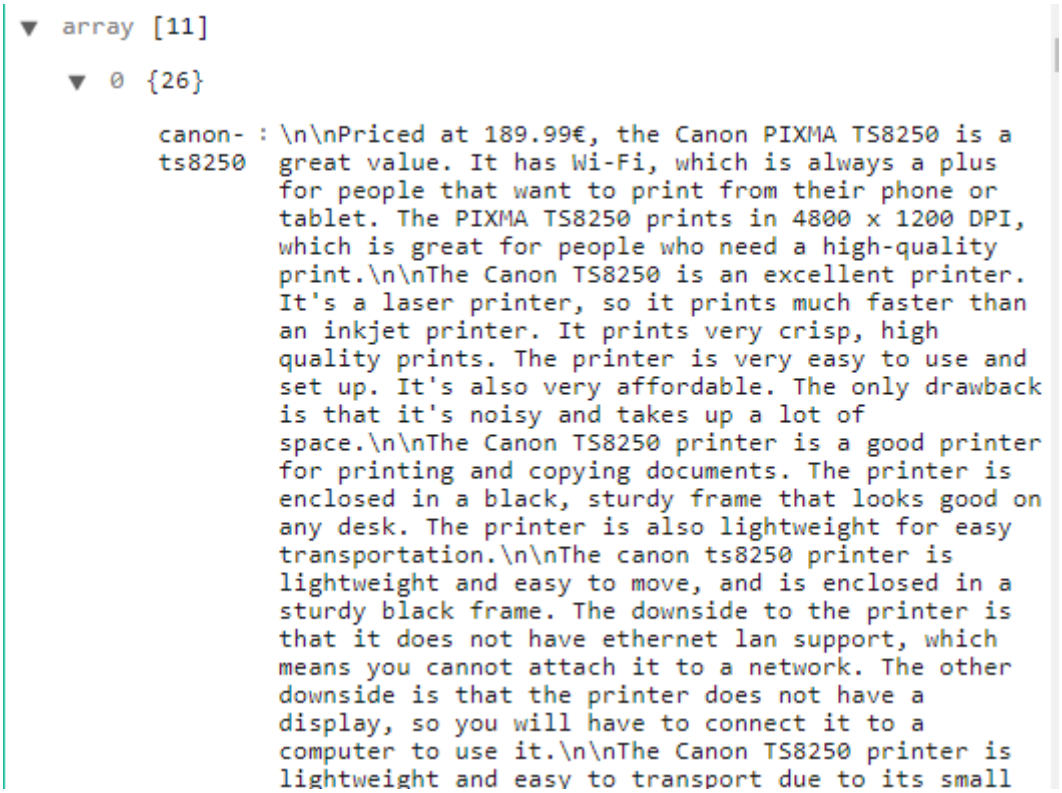

Figura 6.9 Esempio di descrizione prodotto rete GPT-3 in formato JSON

I dati saranno memorizzati all'interno del file in modo che la chiave corrisponda al nome del dispositivo, mentre il valore corrisponda alla descrizione del prodotto. Nella Figura 6.9 si può notare come venga organizzato il file di output.

Capitolo 7

Risultati sperimentali

Una parte molto importante di un progetto riguarda lo studio dei risultati che si è ottenuto. Infatti, bisogna trovare un modo di valutare le prestazioni ottenute in maniera empirica. Per valutare le prestazioni esistono vari modi, la maggior parte dei quali non è utilizzabile nel nostro caso. Nella letteratura del NLP i metodi più usati per misurare le performance di una rete neurale sono le misure (Brownlee, 2017)BLEU e (htt)ROUGE, utilizzate da sempre per valutare task legati alla generazione di testo in modalità supervised, ma inutili nel caso di task di tipo unsupervised. Per esempio, il BLEU viene utilizzato nell'ambito delle traduzioni da una lingua ad un'altra, e l'algoritmo non fa altro che confrontare la traduzione generata dalla rete con la stessa traduzione generata in un altro modo, e verificare quanto esse siano simili.

Il seguente capitolo sarà incentrato sulle tecniche utilizzate per misurare le performace del sistema e per valutare sia in maniera analitica sia visiva quale rete sia la migliore nella generazione di testo in maniera automatica effettuando dapprima una analisi basata sulla misura LSA, ed in seguito una analisi manuale sia dei testi generati con GPT-2 sia con GPT-3 in modo da riuscire a valutare al meglio entrambi i modelli.

7.1 Latent Semantic Analysis

Latent Semantic Analysis, abbreviato (Ioana, 2020) **LSA**, è una tecnica di misura usata nell'ambito del Natural Language Processing e ha come scopo primario quello di riuscire ad estrarre dei pattern dal testo e di determinare se esso è coerente oppure no. Generalmente viene usata nell'ambito di text summarization e text classification, riuscendo in pratica ad ottenere una riduzione di dimensionalità attraverso **SVD** (Singular Value Decomposition).

Il concetto su cui si basa è quello di scomporre un testo in parole elementari. Si supponga di avere n documenti, ognuno di essi composti da un determinato numero di parole. Si può definire una tabella in cui ogni riga corrisponde ad un generico documento, mentre ogni colonna rappresenta la presenza oppure l'assenza di una determinata parola all'interno del documento.

Partendo da tale tabella di dimensione N x d è possibile effettuare una operazione di riduzione della dimensionalità della tabella. Per effettuare tale step è necessario definire un insieme di "Topic", che corrispondono a delle categorie a cui un determinato documento può appartenere.

Sarà definita una colonna per ogni topic diverso, ed in generale si dovrebbe avere che il numero di topic C sia minore del numero di parole presenti all'interno del vocabolario.

$$c \leq d$$

Supponiamo che tale condizione sia verificata, adesso è possibile definire una nuova matrice avente N righe e C colonne, in cui il valore di ogni colonna è un valore numerico che specifica in maniera analitica quanto un documento risulta essere collegato ad una determinata categoria. Come è giusto che sia possono esservi determinati documenti che possono appartenere allo stesso tempo a più categorie, ed in quel caso varrà la regola secondo cui si prenderà come categoria principale il valore massimo. Con il seguente approccio avremo ridotto la dimensione della tabella di partenza. Il tutto usando SVD, che ci consente di poter rappresentare una matrice attraverso un'operazione di fattorizzazione in tre matrici distinte, come mostrato in Figura 7.1.

Se si moltiplicano tali matrici si otterrà la matrice di partenza, attraverso l'uguaglianza:

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

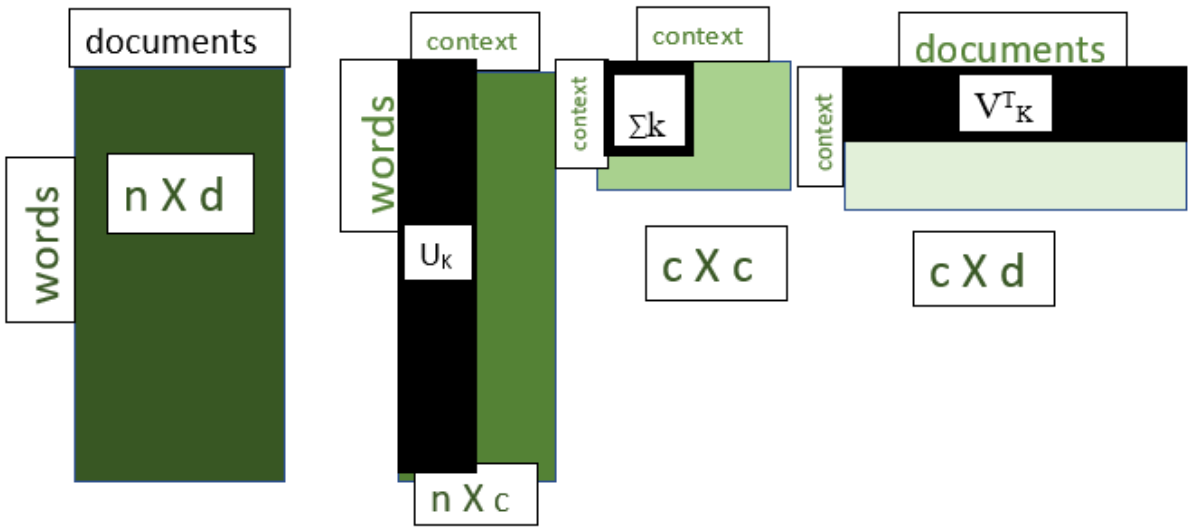

Figura 7.1 Scomposizione matrice documento-parole attraverso fattorizzazione in tre matrici distinte.

Le matrici U e V sono matrici ortonormali mentre la matrice **Σ** è una matrice diagonale; quindi, è progettata in modo che soltanto gli elementi lungo la diagonale principale siano non nulli. I valori presenti all'interno della matrice **Σ** rappresentano quanto ogni concetto latente spiega la varianza nei dati.

7.1.1 Implementazione

Per la sua implementazione esistono numerose librerie che è possibile utilizzare, in particolare in Python, dove ne esistono numerose Open Source che implementano l'algoritmo descritto sopra via software.

In particolare, nel nostro caso si è deciso di utilizzare le librerie "gensim" e "nltk", le quali combinate insieme ci hanno permesso di implementare un modello LSA.

(gensim, 2021)Gensim è una libreria specifica per task legati al NLP, ed in particolare fornisce delle implementazioni molto efficienti di algoritmi legati al topic modeling, nel nostro caso algoritmo LSA.

(ntlk, s.d.)Ntlk, invece, è una libreria Python specifica per lavorare con dati testuali, in particolare viene utilizzata per effettuare tokenization, stemming e per tanti altri scopi legati al processing di dati testuali.

Anche nell'implementazione di un algoritmo che faccia operazioni legate al topic modeling si deve definire una pipeline, definendo in modo opportuno come le varie operazioni sono fatte, e soprattutto in quale ordine.

7.1.2 Pipeline

Ci si concentri ora sulla pipeline, definendo il flusso di operazioni necessarie per generare la misura desiderata applicando l'algoritmo LSA.

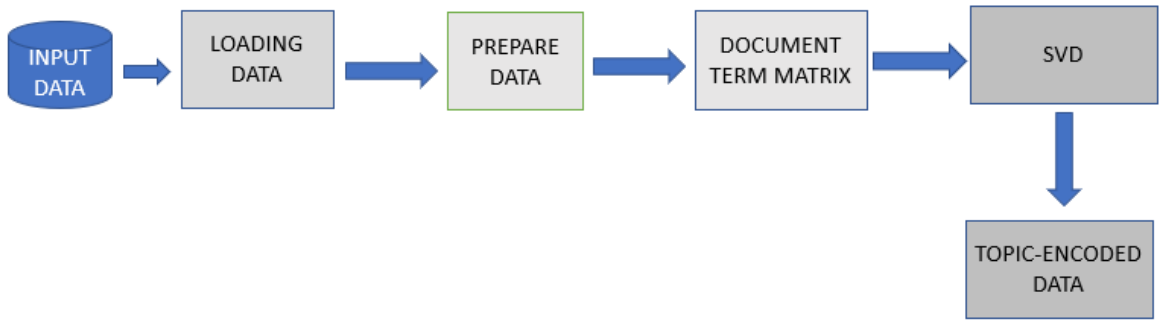

Figura 7.2 Pipeline implementazione algoritmo LSA

Per comodità è stata riassunta in Figura 7.2, laddove sono state definite in ordine tutte le varie fasi e come esse sono collegate tra loro. Vediamole ora nel dettaglio. I dati di partenza saranno quelli ottenuti come output dalla rete neurale. Lo scopo, infatti, qui è quello di voler vedere quanto i documenti generati siano "buoni". I dati sono stati in precedenza memorizzati in formato JSON, in modo che ad ogni documento JSON corrisponda un documento testuale generato dalla rete.

Nel primo step della pipeline si effettua una lettura da file, ed i dati verranno memorizzati in RAM all'interno di una variabile.

```
for index in range(0,len(data)):
        text = data[index].strip()
        documents_list.append(text)
```

Tabella 7.1 Loading dei documenti contenuti nel file.

I dati presenti all'interno del file vengono visti come un vettore, ragion per cui si effettua una iterazione attraverso cui i vari documenti verranno inseriti all'interno di documents_list dopo averci applicato l'operatore strip () sulla stringa passata come parametro. Al termine di tale step sarà ritornata una lista contenente i vari documenti convertiti in stringhe.

Subito dopo tale fase i dati devono essere pre-processati. Il pre-processing avrà come obiettivo quello di dividere il testo in token, rimuovere le stopword ed infine effettuare lo stemming. Per effettuare la divisione in token si è deciso di utilizzare l'oggetto `RegexpTokenizer` dalla libreria ntlk, il quale riceve come parametro una espressione regolare, la quale verrà usata per la divisione in token del testo di partenza. In particolare, si è deciso di voler effettuare una divisione basata sulle word, quindi ogni token corrisponderà ad una parola presente nel testo. Si inizializza il set delle stopwords con l'elenco di tutte le stopwords presenti all'interno della lingua inglese. Per fortuna tale operazione viene fatta in maniera automatica dal sistema, dal momento che in rete è presente una lista contenente al suo interno tutte quelle parole.

```
nltk.download('stopwords')
en_stop = set(stopwords.words('english'))
```

Tabella 7.2 Utilizzo degli stopwords con la libreria nltk.

La prima istruzione è stata utilizzata per scaricare la lista delle stopwords da internet, mentre la seconda istruzione è stata usata per inizializzare la lista delle stopwords a partire dalla risorsa scaricata in precedenza. Infine, si inizializzerà uno stemmer con `p_stemmer = PorterStemmer ()`. Lo stemmer serve in quanto ci consente di ricondurre parole derivate alla loro radice. Per esempio, nel caso in cui ci siano verbi coniugati essi saranno ricondotti tutti alla versione base, in modo che vengano tutti trattati allo stesso modo e non come entità separate. Dopo aver inizializzato il tutto si itera all'interno di un ciclo for attraverso tutti i documenti. Ogni documento viene prima convertito in lowercase, e solo successivamente si applicano in sequenza le operazioni descritte in precedenza.

```
texts = []
    for i in doc_set:
        raw = i.lower()
        tokens = tokenizer.tokenize(raw)
        stopped_tokens = [i for i in tokens if not i in en_stop]
        stemmed_tokens = [p_stemmer.stem(i) for i in stopped_tokens]
        texts.append(stemmed_tokens)
```

Tabella 7.3 Text pre-processing utilizzando ntlk, con dettaglio sulla tokenization e stemming.

La funzione tokenize applicata alla stringa di partenza ci ritornerà una lista di token, la quale verrà opportunamente filtrata tramite le funzioni lambda definite all'interno delle righe successive. Infine, verrà ritornato come output il testo pre-processato.

Si deve generare ora la document-term matrix, ovvero la matrice in cui ogni riga corrisponde ad un documento ed ogni colonna corrisponde ad una parola distinta. Tale step viene implementato tramite la generazione di un dictionary, laddove ad ogni parola unique viene assegnato un valore numerico univoco. Successivamente, a partire da tale dizionario viene generata la matrice utilizzando il metodo doc2bow applicato al dictionary prodotto in precedenza, che consente di trasformare un documento in una bag of words. Il tutto viene formalizzato con le seguenti righe di codice:

```
dictionary = corpora.Dictionary(doc_clean)
doc_term_matrix = [dictionary.doc2bow(doc) for doc in doc_clean]
return dictionary,doc_term_matrix
```

Tabella 7.4 Codice che genera la document term matrix.

Quindi, per ogni documento all'interno della document_list vi sarà applicata la funzione doc2bow, che produrrà in output una bagOfWords.

Nei passi successivi si dovrà generare il modello, che, come abbiamo detto in precedenza, viene ottenuto applicando l'algoritmo SVD che ci consente di fattorizzare una matrice di partenza per poter ottenere tre matrici, le quali moltiplicate ci ridaranno la matrice di partenza.

`lsamodel = LsiModel(doc_term_matrix, num_topics=number_of_topics, id2word = dictionary)` ci produrrà in output il modello LSA che useremo. Da come si vede, si deve specificare il numero di topic che si vuole usare, così che si potrà generare il modello in maniera opportuna. La sfida consiste nel riuscire a trovare il giusto valore di numero di topic, che deve essere adeguato al contesto che si sta trattando. Infatti, più esso è grande e più si avranno risultati dettagliati e specifici.

Infine, non si dovrà fare altro che lavorare sul modello prodotto in precedenza; nel nostro caso si è deciso di voler rappresentare in forma visiva utilizzando dei grafici come varia la coerenza al variare del numero di topic. Otterremo, quindi, una misura generale relativa a tutti i documenti che abbiamo prodotto.

```
for num_topics in range(start, stop, step):
        model = LsiModel(doc_term_matrix, num_topics=number_of_topics,
 id2word = dictionary)
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=doc_clean,
dictionary=dictionary, coherence='c_v')
        coherence_values.append(coherencemodel.get_coherence())
```

Tabella 7.5 Codice che genera il grafico in cui l'ascissa rappresenta il numero di topic, le ordinate, invece, il coherence score.

7.2 Esperimento

Dopo aver descritto la pipeline si vuole descrivere come è stato impostato l'esperimento. Si è deciso di valutare le prestazioni del modello generato usando GPT-3 e quelle ottenute usando la rete GPT-2, in cui la prima ottenuta usando il modello unico per le categorie printer e smart-tv, e nel secondo caso usando due reti separate.

Infatti, si deve decidere il valore di numero di topic tramite cui si vuole scomporre il modello. Se si usa un valore molto grande si avrà come risultato un modello che è molto specifico, ma al tempo stesso una matrice molto sparsa, mentre nel caso di valori molto piccoli accade l'esatto opposto, e magari può capitare che alcune informazioni rilevanti vengono scartate.

Il primo esperimento è stato calcolato fissando il numero di topic a 10; di seguito verranno riportati i grafici ottenuti.

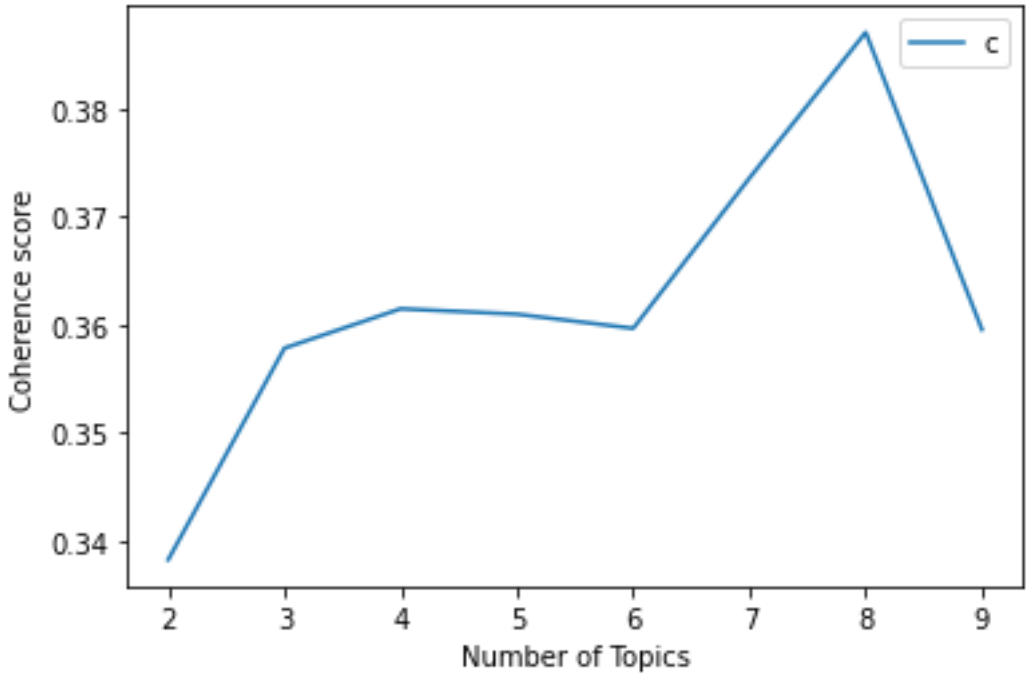

Figura 7.3 Grafico ottenuto dalla rete generica

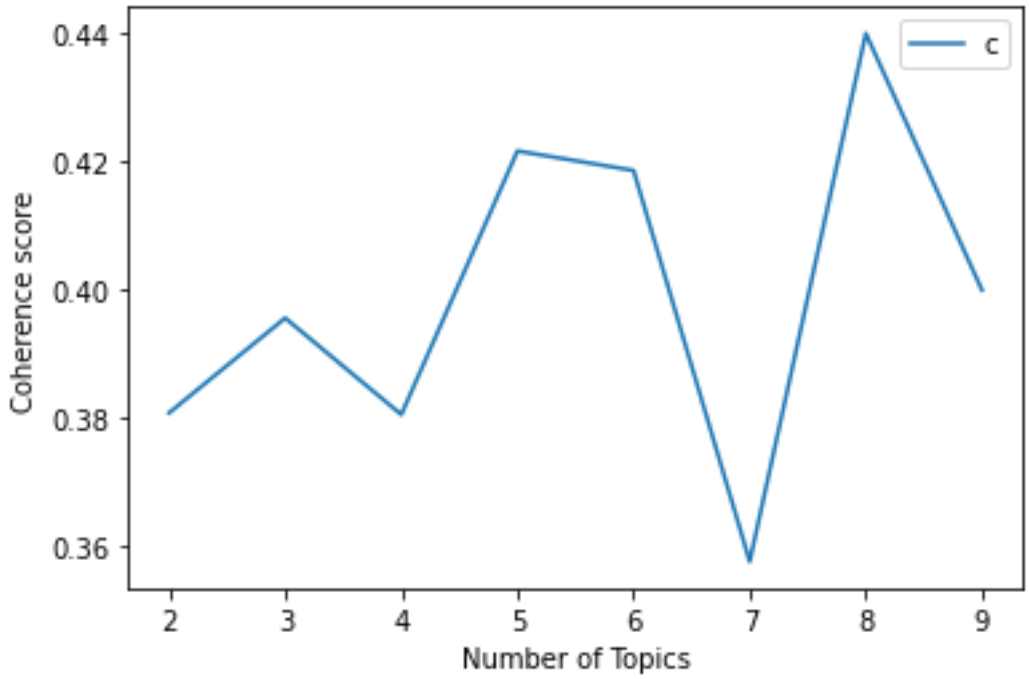

Figura 7.4 Grafico relativo alla rete specifica

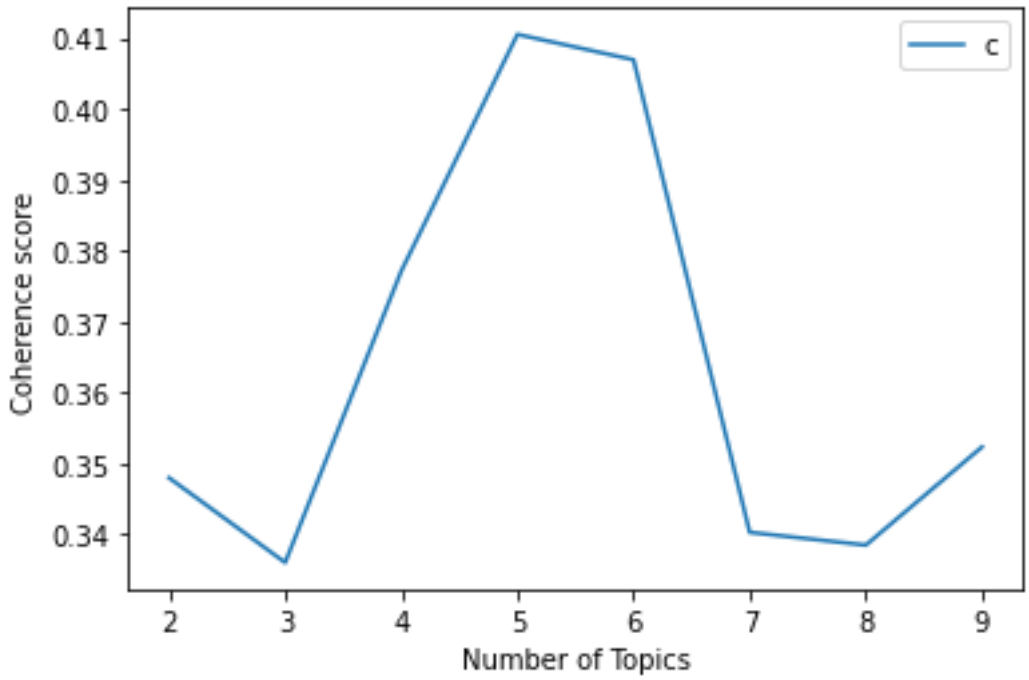

Figura 7.5 Rete GPT-3

I grafici riportano sulle ascisse il numero di topic, e mostrano come varia la coerenza del testo al variare del numero di topic. In questo caso è stato fissato il valore massimo a 10. A primo sguardo si nota subito che le prestazioni migliori si ottengono usando il modello specifico di GPT-2 e la rete GPT-3. La cosa più sorprendente, però, è che i risultati ottenuti con la rete GPT-3 sono stati raggiunti utilizzando una rete pre-trained, che ci mostra quanto sia potente quest'ultimo modello, e con margini di crescita e potenzialità infinite. Analizzando i grafici si può determinare in maniera analitica qual è il valore da dover assegnare al numero di topic per massimizzare la coerenza. Sfortunatamente tale valore non è univoco, come si vede da Figura 7.3,Figura 7.4 e Figura 7.5.

Al crescere del numero di topic si ha una diminuzione della coerenza del testo generato, dal momento che saranno introdotte sempre nuove categorie, ragion per cui lo score associato ad ogni singola categoria calerà bruscamente.

Vediamo a puro scopo dimostrativo il comportamento dei modelli al variare del numero di topic per N=1 ad N=185.

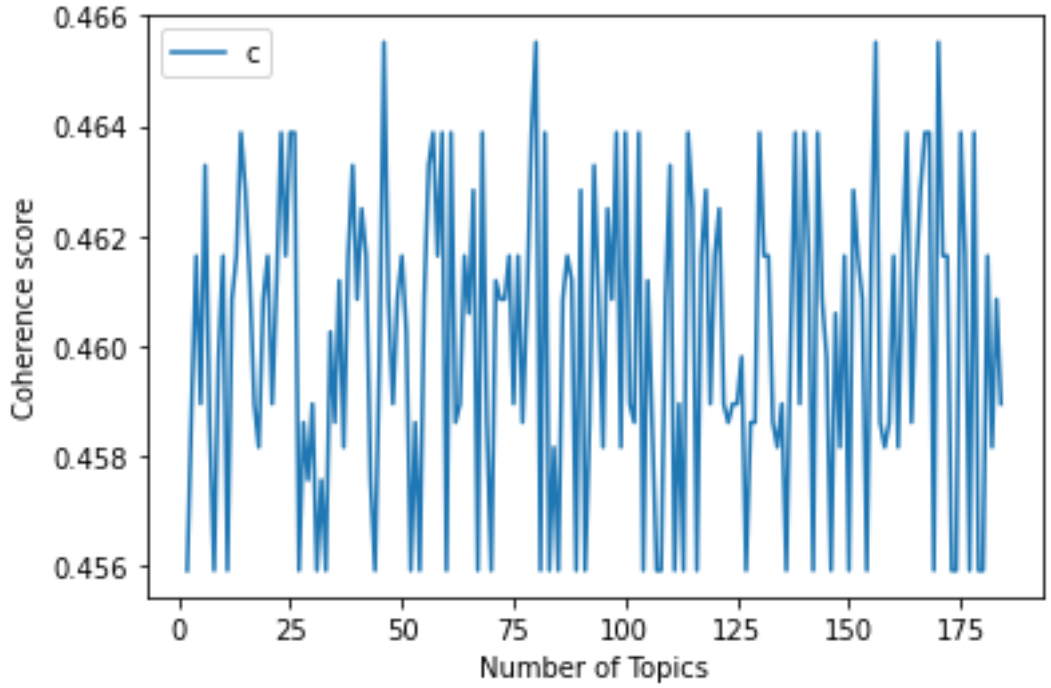

Figura 7.6 Rete GPT-3

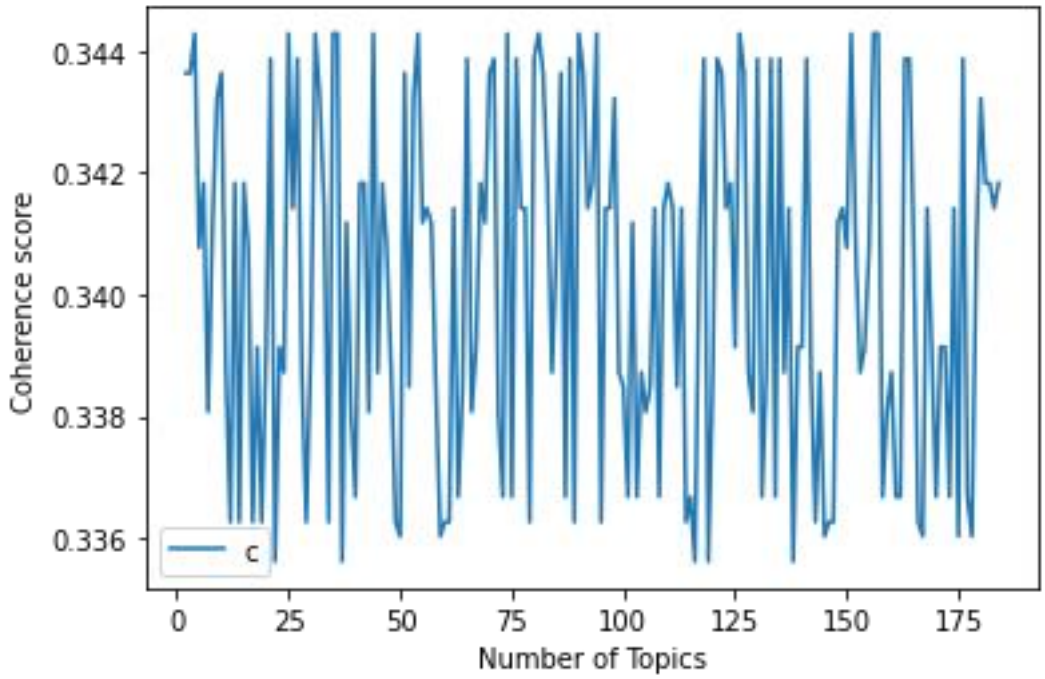

Figura 7.7 Rete GPT-2 unica

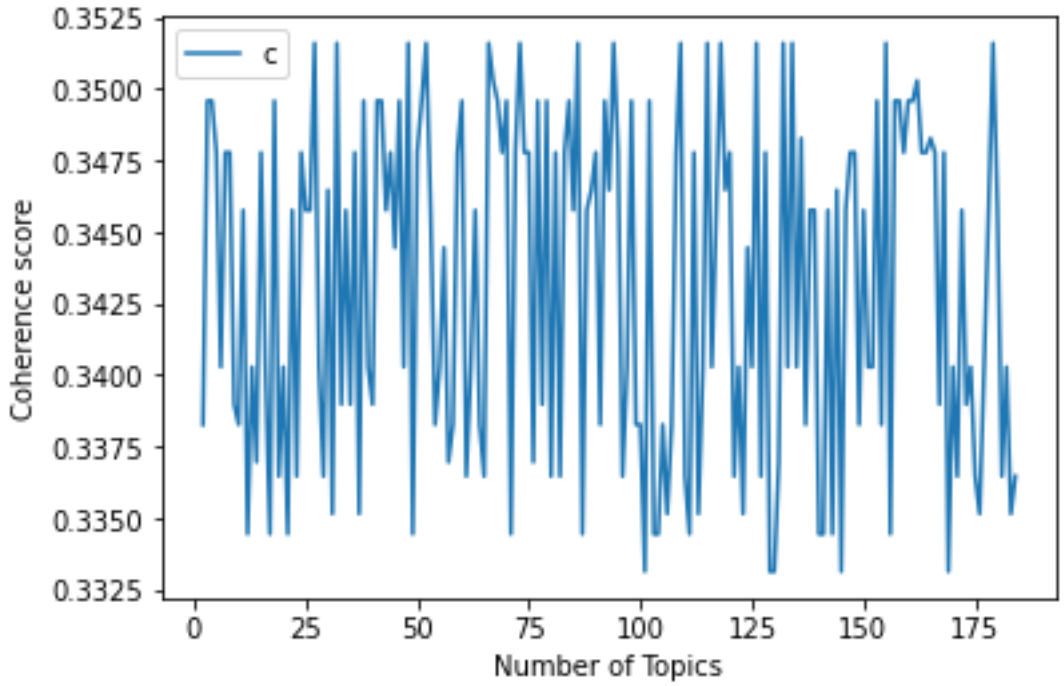

Figura 7.8 Rete GPT-2 separata per N=185

Come si vede dalle figure Figura 7.6, Figura 7.7, Figura 7.8 il grafico avrà un comportamento molto variabile, e soprattutto oscillatorio alternando punti di massimo e minimo in maniera molto casuale e variabili al variare del numero di topic. Di base non è una scelta molto sensata quella di voler lavorare con un numero di topic così

grande, dato che la dimensione della matrice risultante cresce a dismisura, rendendo difficili le operazioni computazionali. Questo è un caso particolare dal momento che si è scelto in maniera opportuna il numero di topic uguale al numero di documenti in input, che in teoria avrebbe potuto generare un topic separato per ogni documento in input, anche se in pratica i vari documenti appartengono a topic separati (che alla fine è quello che vogliamo).

7.3 Valutazione Manuale

Per analisi manuale si intende una vera e propria analisi del testo, effettuata per un subset di documenti generati dalle diverse reti, in modo da vedere in maniera molto accurata se la qualità del testo generata è buona, o se sono presenti determinati problemi che magari non sono visibili attraverso analisi automatiche. I documenti di partenza sono i documenti generati con le tre diverse reti neurali, ed il nostro obiettivo è quello di voler determinare in maniera qualitativa come si comportano le varie reti nella generazione del testo. Ci si concentrerà su un modello di rete alla volta, cercando di analizzare nei dettagli come sono i testi generati.

Il primo modello che considereremo è GPT-2 in configurazione iniziale, quindi quella che è stata addestrata simultaneamente su entrambi i domini. Tale modello soffre di un problema, non proprio irrilevante:

"The television LG oled55gx6la has a price of 1594.44€ for a 4-series OLED. The thickness is of 23.1mm, and the build volume is about 13,000 sheets. I tested the printer over an Ethernet connection with its drivers installed on a system running Windows Vista. The display resolution is of 3840 x 2160 (4K) pixels. That's 2,376.36cd/m 2 and a modest peak brightness of 498. The weight is of 14.2kg, and the included remote is a 9.7-by-11.3-inch black rectangle with a circular navigation pad below it. The ethernet lan rj-45 port is a 1.3-foot port for connecting a single PC to the printer without a computer. The usb 2 0 port is a 2.3-inch-thick, 0.35-pound black plastic sheet that can be stored behind a counter and attached to the keyboard. The hdmi port is a 4.7-inch wand wheel mounted on the lower-left of the display and flanked by buttons."

Si prenda in considerazione la seguente recensione, generata con la rete in questione, e riguardante le Smart-TV. Come si può notare, nonostante essa sia una recensione che riguarda una TV, molto spesso accade che se ci sono degli attributi sovrapposti tra le due categorie di merci, si può ottenere una recensione finale al cui interno possono essere presenti riferimenti ad un dominio che non dovrebbe essere presente in quel momento.

Se ci si concentra sullo stile del testo generato, esso è molto buono, si ispira molto allo stile delle recensioni usate per addestrare la rete, e pertanto esso non è mai macchinoso. L'unico problema si ha nel punto in cui si congiungono due diverse frasi, ma in quel caso si può risolvere aggiungendo variabilità nella funzione che si occupa della generazione del testo.

Si consideri ora un'altra recensione proveniente dalla stessa rete, ma appartenente al dominio Printers, di seguito ne sarà riportato un estratto.

"The printer HP deskjet plus 4130 has a price of 59.98€, just slightly lower than the Officejet 7740, but lower than either. The print speed colour is a 5.5ppm, which isn't much of a difference at all. As I mentioned above, the MB472w is not a full-blown monochrome laser. The printer features is a Contains More Than 20 % Recycled Plastic. It measures 1.5 inches thick, is covered in clear plastic, and, of the plastic that comes with the printer, plastic that's recycled is 99. The dimensions wxdxh is of 200 x 428 x 332 mm (H x W x D)mm and the square height is of 0.12m. The standard interfaces is a Usb, Usb 2x2, and Direct. However, the OfficeJet Pro 6978-J holds an inferior sibling to the Pro-1000 for one of the first real interfaces to a color laser printer. The weight is a 4.8kg monochrome monochrome laser that can be mounted on a wall or in a recessed area. The minimum system requirements is a Windows 7 & 10 - MacOS 10.12 (Sierra) or Later. However, the printer must have a Wi-Fi access point on site for connecting to it."

Un altro problema riscontrato nella rete GPT-2, indipendentemente dalla versione, è stato evidenziato nel testo. Nella seguente recensione si sta parlando della stampante HP deskjet plus 4130, ma come si nota nel testo, la rete finirà con il parlare di tutte le altre stampanti che conosce, OfficeJet Pro 6978-J e MB472w le quali come si può intuire facilmente differiscono totalmente da quella iniziale. Allo stesso modo, la stessa cosa si verifica anche nel caso di valori numerici, come può essere il prezzo della stampante, oppure il suo peso, che magari prima assumerà un valore e poi subito dopo un altro.Per ridurre la possibilità che possano essere presenti errori di questo tipo all'interno del testo generato si è deciso di impostare come parametro per la generazione di testo un valore di numero token di output relativamente piccolo, in modo che la rete non abbia molto modo di effettuare delle digressioni in cui mettersi a parlare di altro.

Consideriamo adesso il modello successivo. Si vuole vedere se il problema che avevamo in precedenza è stato risolto dalla nuova rete oppure no.

The television LG oled55bx6lb has a price of 1098€ and a peak brightness of 501cd/m 2 for a 42,715:1 contrast ratio. The display diagonal is of 55" height and weighs 60 pounds. The thickness is of

62mm, along with a strip of bar tape. This is about the same thickness as the tape, plus the two standard length strips. The high frequency rate is of 60Hz, which is a good setting for gaming. The Low Frequency Setting gives you the same performance. The display resolution is of 3840 x 2160 (4K) pixels. The speaker is a 2.1-inch wide aluminum speakerbar with a touchpad and clickable scroll wheel. The video apps is a Amazon prime video, Foxxum, Netflix, Youtube, Spotify, Skype, and YouTube. It has smart tv and other smart features like Netflix, Hulu Plus, Facebook, and Twitter and is compatible with Android smartphones and tablets. The sound enhancement technology is a DTS Studio Sound - Dolby Audio, which helps to create a deeper, more muted sound without appearing slightly garish. The power consumption typical is of 150W, and the M55Q6 gets 75 watts. At 120Hz (which uses 0.5C), the screen consumes 128 watts of power. The weight is of 13kg with a two-year warranty, and the screen is rated at 75 inches long. The ethernet lan rj-45 port is a 1.3ms lag, with an additional 1ms input lag at the start of the game. The usb 2 0 port is a 2.5mm thin (HWD) silver band that sticks to almost any surface when closed. The hdmi port is a 3.5mm connector that plugs into the port on top of the box, into the left side of the back of the HDTV.

Leggendo la seguente recensione ci si accorge subito che non compaiono più all'interno del testo termini come printer e parole affini, ma compariranno soltanto termini che si riferiscono al mondo delle televisioni. Anche in questo caso lo stile della recensione è molto sorprendente, dato che conserva lo stile narrativo che avevano le recensioni originali. Ancora una volta è presente il problema descritto in precedenza, anche se in questo caso si verifica con minore frequenza rispetto al caso precedente.

Ovviamente tale problema tende a verificarsi con maggior frequenza in maniera proporzionale alla lunghezza dei periodi che si genererà. Ancora una volta sarà presente una discontinuità tra le varie frasi differenti, e poca varianza nelle frasi usate come connettore tra i diversi periodi, ma in questo caso la rete non ha nessuna colpa dal momento che quella è la frase iniziale inviata come input alla rete per la generazione del testo, e generata in maniera automatica da noi. Quindi, come detto in precedenza, si potrebbe lavorare per cercare di affinare quel processo.

"The printer HP deskjet plus 4130 has a price of 59.98€ for a single sheet, and the OKI MC562w ($119.98 at Amazon) is 50.98€. The print speed colour is a 5.5ppm, which matches the Canon B2370d's of 6.2ppm and the Canon MF232w's 5.7ppm. The Canon MF251dw's 5. The print technology is a Inkjet rather than a printing press. Print speed in my tests was very fast for an inkjet, with averages of 1 minute 35 seconds for one side and 5 seconds for the other. The dimensions wxdxh is of 200 x 428 x 332 mm (H x W x

D)mm with a 60kg roll thickness of 0.6mm in front and 0.2mm behind the back. The standard interfaces is a Usb, Usb 2.0, or Usb Lite. HP gives you an ePrint and Scan interface and an Ethernet port. The weight is a 4.8kg, but the only paper you can move is a few sheets in each tray on top. The minimum system requirements is a Windows 7 & 10 - MacOS 10.12 (Sierra) or Later system with automatic duplex printing. It's also compatible with the following macOS and Linux systems: macOS Lion, macOS 10."

Come di quanto detto in precedenza è stato riportata anche la recensione della HP deskjet plus 4130 generata con questa rete. Ancora una volta è presente il problema che ho più volte descritto, relativo al cambio di valori degli attributi, ed inoltre nel seguente caso particolare la lunghezza complessiva del testo è maggiore rispetto quella della recensione precedente. Probabilmente sarà stato generato del testo per qualche campo aggiuntivo rispetto a quella precedente.

La cosa più sorprendente di modelli basati su GPT-2 è la capacità di emulare testi scritti da esseri umani, riuscendone a copiare lo stile e la tecnica con elevata fedeltà, anche se da un punto di vista semantico la qualità delle recensioni generate non è molto accurata, per tutta quella serie di problemi descritti nell'arco di questo capitolo.

Concentriamoci ora sulla rete GPT-3. Anche in questo caso si mostrerà un esempio di testo generato con il seguente modello di rete, e si cercherà di fare una analisi del contenuto. Ancora una volta ci soffermeremo sulla categoria delle stampanti per effettuare il confronto tra le due diverse reti.

"\n\nThis printer is a great value for the price. It is a small, sleek printer that is simple to set up. The only complaint I have is that the paper tray is small, but I am okay with that because that means I am less likely to run out of paper. The print quality is excellent and there are many features that make this printer stand out. The print speed is fast, the print quality is excellent, and it is very energy efficient. This printer is perfect for a small home office or for a student. It is also perfect for a home because it is wireless.\n\nHP Deskjet Plus 4130 All-in-One Wireless Inkjet Printer - HP 305 Black Ink Set-Up Cartridge - HP 305 Tri-Colour Ink Set-Up Cartridge - Power Cable - Set-Up Instructions, wireless technology Yes, type Home Printer\n\nThe HP deskjet plus 4130 is a good printer for home and small business use and is the perfect size for a home office. It is a very streamlined printer with just what you need, and nothing more. The HP deskjet plus 4130 prints really fast and is very easy to use.\n\nThe HP deskjet plus 4130 is a good printer for home and small business use and is the perfect size for a home office. It is a very streamlined printer with just what you need, and nothing more. The HP deskjet plus 4130 prints really fast and is very easy to use. The HP deskjet plus 4130 is not as expensive as

some printers, but it doesn't have all the features of the more expensive printers.\n\nThe HP deskjet plus 4130 is not as expensive as some printers, but it doesn't have all the features of the more expensive printers. The HP deskjet plus 4130 is a very sleek and streamlined printer. It is simple to set up and use, and it prints quickly. The HP deskjet plus 4130 is the perfect size for a home office and it doesn't take up a lot of space. The HP deskjet plus 4130 can print in both black and white and color.\n\nThe HP deskjet plus 4130 is a fast, wireless printer that can print a variety of sizes. It has a large paper capacity of 60 sheets and can print up to 12 pages per minute. It has many different input trays that can hold a variety of paper sizes. It also has a large LCD screen and is easy to setup.\n\nThe HP deskjet plus 4130 is a great printer for anyone who wants a reliable machine that can handle a variety of printing needs.\n\nThe HP Deskjet Plus 4130 is a printer that offers a combination of speed and affordability with excellent photo quality. Though it is not a wireless printer, it is very easy to setup and operate. It is a great printer for a home or small business office.\n\nThe HP Deskjet Plus 4130 is a great printer for a home or small business office. The printer offers a combination of speed and affordability with excellent photo quality. Though it is not a wireless printer, it is very easy to setup and operate."

Il testo generato risulta essere molto più coerente semanticamente rispetto quello generato con la rete precedente, effettivamente si parla per tutto l'articolo della stampante hp deskjet plus 4130. Forse si parla anche troppo della stampante, viene menzionata all'inizio di ogni nuovo periodo dalla rete, e questo è un problema non di poco conto, dal momento che un lettore esterno potrebbe annoiarsi nel leggere ripetutamente la stessa cosa. Tralasciamo per un attimo questo aspetto. Una cosa sorprendente è che la rete è in grado di generare da sola la recensione senza doverle fornire una frase iniziale, ed infatti ogni recensione diversa presenta un inizio del testo differente.

Lo stile narrativo è uno stile tipicamente usato quando si vuole fare una recensione scritta a mano, anche se i punti maggiormente stonati sono quelli in cui si vuole forzare la rete nell'inserire determinati riferimenti testuali, in cui non si farà altro che ripetere gli stessi concetti numerose volte, come nel caso riportato sopra, in cui viene riportate alcune volte la frase in cui si parla della velocità di stampa e della facilità d'uso della stampante. Non sono presenti valori numerici nel testo, ma essi sono sostituiti da giudizi su una determinata feature, in cui per esempio si elogia una determinata caratteristica oppure un'altra. Una recensione dovrebbe contenere sia una descrizione delle specifiche di un prodotto sia un parere riguardo l'oggetto in questione, mentre in questo caso la parte descrittiva non viene considerata.

Capitolo 8

Conclusioni e sviluppi futuri

È arrivato il momento di trarre una conclusione su quanto è stato fatto. Il core dell'applicazione è stato implementato utilizzando diversi approcci, e valutando i risultati si riesce già a notare la differenza tra i vari modelli utilizzati.

L'utilizzo di Google Colaboratory come ambiente di sviluppo ha condizionato moltissimo le scelte implementative che sono state fatte. Infatti, è stato necessario scendere a compromessi, utilizzare dei modelli non troppo grandi e dei dataset di dimensioni ragionevoli. Per esempio, basti pensare alla scelta della rete, laddove si è stati vincolati nel poter usare o la versione small oppure la versione medium di GPT-2, dal momento che le rispettive versioni Large ed Extra-Large non riuscirebbero ad essere eseguite su Colab a causa della RAM insufficiente. Si è dovuto prestare attenzione a far durare i vari task non più di dodici ore, e di salvare tutto in remoto per poter accedere in maniera efficiente ai dati.

La rete implementata usando GPT-2 riesce a generare del testo che presenta un tono espressivo simile a quello dei testi passati come input alla rete; quindi, presenta il vantaggio che è simile al testo scritto da degli esseri umani che hanno scritto quelle recensioni. Di contro, però, la rete funziona bene e genera delle buone recensioni fin tanto che la rete è stata addestrata sia su un numero sufficiente di articoli sia su quel determinato articolo di cui si vuole generare una descrizione. In caso contrario verranno generate recensioni al cui interno si noterà un repentino cambio di contesto da un articolo ad un altro.

La rete ottenuta usando GPT-3, invece, non è ancora addestrabile, può essere usata soltanto in versione demo attualmente, però produce dei risultati che dal punto di vista semantico sono notevoli. Il testo generato è sempre coerente anche se risulta essere molto ripetitivo. Per quel che riguarda GPT-3, tuttavia, non può esser fatto molto per cercare di migliorare la qualità del testo generato, almeno per ora. Probabilmente il vantaggio principale di GPT-3 è che il tutto viene gestito tramite opportune API, vi è un server esterno che si occupa di generare il testo, e gli scambi di dati avvengono tramite protocollo HTTPS. Non vi è bisogno di acquistare alcun dispositivo fisico per il seguente scopo, ma il tutto può esser fatto semplicemente pagando OpenAI per ricevere un servizio.

Riassumendo, quindi, da una parte abbiamo GPT-2, che può essere migliorata per ottenere delle prestazioni migliori, e dall'altra parte vi è GPT-3 che ha delle prestazioni buone in ogni ambito, ma che non è personalizzabile. La scelta che deve essere fatta riguarda soprattutto il tipo di utilizzo che ne si vuole fare. Se si vuole una rete plug and play probabilmente GPT-3 è la scelta migliore, in caso contrario GPT-2 è un buon compromesso.

Eventuali sviluppi futuri del seguente elaborato di tesi coinvolgono un inserimento dell'elaborato all'interno di un contesto più vasto, che può essere un suo utilizzo per generare come output delle pagine web anziché dei documenti scritti in formato JSON. Un altro sviluppo futuro può riguardare l'ampliamento del numero di categorie gestibili, che in questo caso è fermo a due categorie, oppure modificare la funzione generatrice del testo che viene passato come input alla rete neurale.

Bibliografia

(s.d.). Tratto da https://www.ccs.neu.edu/home/vip/teach/DMcourse/5_topicmodel_summ/notes_slides/What-is-ROUGE.pdf

(2021, 11 16). Tratto da scikit-learn.org: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

A. Radford, J. W. (2019). *Language Models are Unsupervised Multitask Learners*. OpenAI Blog.

Alec Radford, J. W. (s.d.). *Language Models are Unsupervised Multitask Learners*. openai.

blog, C. (2015, Agosto 27). Understanding LSTM Networks.

Brownlee, J. (2017, 11 20). *A Gentle Introduction to Calculating the BLEU Score for Text in Python*. Tratto da https://machinelearningmastery.com/calculate-bleu-score-for-text-python/

Diederik P. Kingma, J. B. (2014). Adam: A Method for Stochastic Optimization.

Dittmar, G. (2021, 01). *towardsdatascience*. Tratto da towardsdatascience.com: https://towardsdatascience.com/natural-language-generation-part-2-gpt-2-and-huggingface-f3acb35bc86a

gensim. (2021, 11 19). Tratto da https://pypi.org/project/gensim/

gitHub. (s.d.). Tratto da https://github.com/simplenlg/simplenlg

huggingface. (2021, 11 16). Tratto da https://huggingface.co/docs/datasets/quickstart.html

Ioana. (2020, 10). *Latent Semantic Analysis: intuition, math, implementation*. Tratto da towardsdatascience: https://towardsdatascience.com/latent-semantic-analysis-intuition-math-implementation-a194aff870f8

Jacob Devlin, M.-W. C. (s.d.). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* .

Joshi, N. (2021, 2 11). *https://www.bbntimes.com/*. Tratto il giorno 10 23, 2021 da https://www.bbntimes.com/technology/3-popular-applications-of-natural-language-generation

Melotti, E. (2016, 11 7). *html*. Tratto da html.it: https://www.html.it/pag/15608/perche-usare-python/

ntlk. (s.d.). Tratto da https://www.nltk.org/

Polosukhin, A. V. (2017). *Attention Is All You Need*. arXiv.

Rewon Child, S. G. (2019, 4 23). Generating Long Sequences with Sparse Transformers.

Rico Sennrich, B. H. (2015, 8 31). Neural Machine Translation of Rare Words with Subword Units. arXiv.

scrapy. (2021, 11 3). Tratto da https://docs.scrapy.org/en/latest/

T. B. Brown, B. M. (2020). *Language Models are Few-Shot Learners*. in Proc. NeurIPS.

Toutanova, J. D.-W. (s.d.). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding .

w3school. (2021, 11 03). Tratto da https://www.w3schools.com/xml/xpath_syntax.asp

Wigmore, I. (2021, 10 21). Tratto da searchenterpriseai.techtarget.com: https://searchenterpriseai.techtarget.com/definition/natural-language-generation-NLG#:~:text=What%20is%20natural%20language%20generation%20%28NLG%29%3F%20Natural%20language,%29%20and%20natural%20language%20understanding%20%28%20NLU%20%29.