ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

SVILUPPO DI RETRIEVAL-BASED CHATBOT PER L'ITALIANO CON TRANSFORMER

Elaborato in
Programmazione di Applicazioni Data Intensive

Relatore
Prof. Gianluca Moro

Co-relatore
Dott. Luca Ragazzi

Presentata da
Luca Morlino

Seconda Sessione di Laurea
Anno Accademico 2021 – 2022

PAROLE CHIAVE

Natural Language Processing

Semantic Textual Similarity

Chatbot

Sentence-BERT

Deep Learning

Work hard. Be nice. Get lucky.

Abstract

L'Intelligenza Artificiale negli ultimi anni sta plasmando il futuro dell'umanità in quasi tutti i settori. È già il motore principale di diverse tecnologie emergenti come i big data, la robotica e l'IoT e continuerà ad agire come innovatore tecnologico nel futuro prossimo. Le recenti scoperte e migliorie sia nel campo dell'hardware che in quello matematico hanno migliorato l'efficienza e ridotto i tempi di esecuzione dei software. È in questo contesto che sta evolvendo anche il Natural Language Processing (NLP), un ramo dell'Intelligenza Artificiale che studia il modo in cui fornire ai computer l'abilità di comprendere un testo scritto o parlato allo stesso modo in cui lo farebbe un essere umano. Le ambiguità che distinguono la lingua naturale dalle altre rendono ardui gli studi in questo settore. Molti dei recenti sviluppi algoritmici su NLP si basano su tecnologie inventate decenni fa. La ricerca in questo settore è quindi in continua evoluzione. Questa tesi si pone l'obiettivo di sviluppare la logica di una chatbot help-desk per un'azienda privata. Lo scopo è, sottoposta una domanda da parte di un utente, restituire la risposta associata presente in una collezione domande-risposte. Il problema che questa tesi affronta è sviluppare un modello di NLP in grado di comprendere il significato semantico delle domande in input, poiché esse possono essere formulate in molteplici modi, preservando il contenuto semantico a discapito della sintassi. A causa delle ridotte dimensioni del dataset italiano proprietario su cui testare il modello chatbot, sono state eseguite molteplici sperimentazioni su un ulteriore dataset italiano con task affine. Attraverso diversi approcci di addestramento, tra cui apprendimento metrico, sono state raggiunte alte accuratezze sulle più comuni metriche di valutazione, confermando le capacità del modello proposto e sviluppato.

Indice

Elenco delle figure

Capitolo 1

Machine Learning

1.1 Che cos'è il Machine Learning

Non si può parlare di Machine Learning senza parlare di Arthur Samuel (1901-1990), informatico americano pioniere dell'Intelligenza Artificiale che, oltre ad aver inventato la tecnica dell'hashing e ad aver spinto all'uso dei transistor nei computer, è accreditato per aver coniato il termine **Machine Learning** (apprendimento automatico) con la sua ricerca sul gioco della dama. Creò il primo programma di dama per il primo computer dell'IBM in commercio: l'IBM 701. Il suo programma è riconosciuto come il primo programma di auto apprendimento per computer al mondo [1]. Fu il primo a parlare di Machine Learning in un saggio considerato una delle pietre miliari dell'Intelligenza Artificiale [2]. In esso descrive il Machine Learning come il campo di studi che offre ai computer la possibilità di apprendere senza essere programmati in modo esplicito.

Se l'Intelligenza Artificiale è la scienza che studia come una macchina possa simulare specifiche capacità dell'essere umano, il Machine Learning è una branca dell'IA che insegna a una macchina come imparare. Il risultato di questo processo è il *modello*. Un modello di ML è un file che è stato addestrato a riconoscere dei pattern. Lo si addestra su un insieme di dati, il *training set*, e gli si fornisce un algoritmo per ragionare e imparare da quei dati. Una volta addestrato il modello, lo si utilizza su dati ed esempi che non ha mai visto prima per fare predizioni su di essi. Le sue applicazioni al giorno d'oggi sono innumerevoli.

1.2 Il Machine Learning oggi

Tradizionalmente gli esseri umani hanno analizzato i dati e i loro cambiamenti ricercando all'interno di essi strutture che li aiutassero a capire e prevedere i loro comportamenti. Al giorno d'oggi, l'esplosione dei dati rende questo studio impossibile e sono quindi i sistemi automatizzati ad interpretare le loro mutazioni.

Il ML circonda la vita di tutti i giorni più di quanto lo si possa immaginare. Se è chiaro che dietro il tagging di oggetti e persone nelle immagini ci sia un modello, non lo è anche per i sistemi di raccomandazione (come su Amazon e Netflix) o per i motori di ricerca. Ogni volta che un utente utilizza Google, usa un sistema che si appoggia su diversi modelli di ML, dalla comprensione del testo della query, all'ordinamento dei risultati in base ai suoi interessi. Altri campi di applicazione sono:

- Riconoscimento vocale
- Classificazione di immagini
- Riconoscimento gestuale
- Rilevamento e prevenzione di frodi
- Sistemi di raccomandazione
- Veicoli autonomi, droni, robotica
- Sistemi di traduzione da una lingua ad un'altra

E tanti altri esempi negli ambiti di sanità, banche, assicurazioni, meteorologia, finanza.

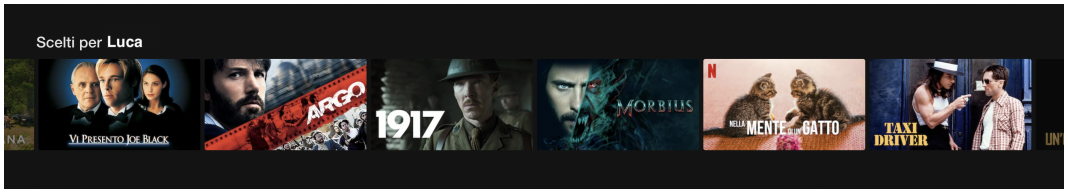

Figura 1.1: Esempio di raccomandazione su Netflix

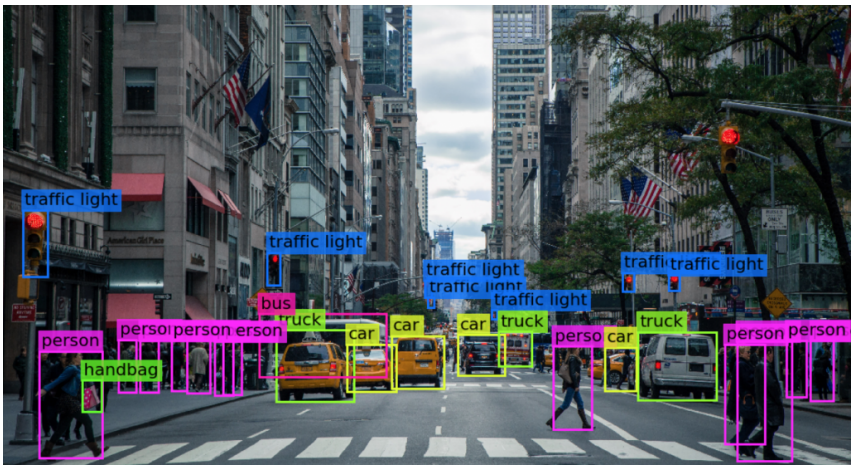

Figura 1.2: Esempio di riconoscimento di oggetti
Fonte: https://miro.medium.com/max/1400/1*wUumH_GH2G82WyqZjROaIQ.png

1.3 Sistemi di Machine Learning

Tra i diversi tipi di sistemi di Machine Learning distinguibili in base alla tipologia di dati con cui vengono addestrati ci sono:

- Apprendimento supervisionato
- Apprendimento non supervisionato
- Apprendimento per rinforzo

1.3.1 Apprendimento supervisionato

Nell'apprendimento supervisionato il training set comprende anche gli output ideali (*labels*) per ogni istanza. Un tipico task di apprendimento supervisionato è la classificazione. Una delle sue più famose applicazioni è il filtro anti-spam delle email (figura 1.3). Il modello viene addestrato su un training set di email già etichettate con *spam* o *non-spam*.

1.3.2 Apprendimento non supervisionato

Nell'apprendimento non supervisionato (figura 1.4), i dati di traning non sono etichettati. Il sistema cerca di riconoscere autonomamente dei pattern sul training set. Un esempio di tale apprendimento è la profilazione degli utenti di un e-commerce. Il modello può notare che il 50% dei clienti è di genere maschile e ama i gatti, mentre il 30% è under-25 e pratica sport. Questo sistema

di raggruppamento, chiamato *clustering*, è utile per incrementare le vendite attuando specifiche strategie di marketing.

1.3.3 Apprendimento per rinforzo

Nell'apprendimento per rinforzo il sistema, chiamato *agente*, osserva l'ambiente, seleziona ed esegue delle azioni e in base ad esse ottiene delle ricompense o delle penalità. Tale approccio è utilizzato nell'implementazione di algoritmi di alcuni robot, per esempio durante l'apprendimento di movimenti come camminare o correre.

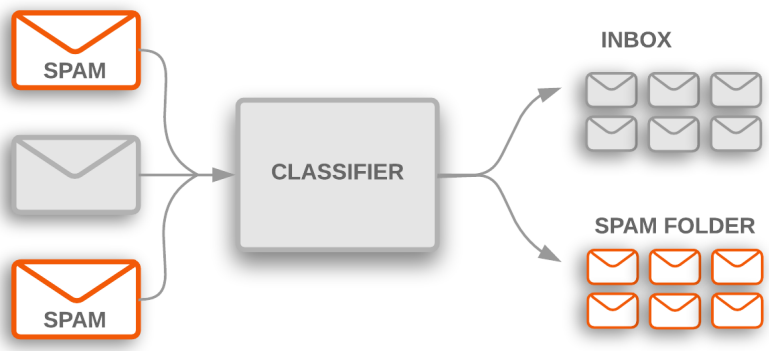

Figura 1.3: Classificatore di spam
Fonte: https://miro.medium.com/max/720/0*qntJ-qbeXfFCh6U7.png

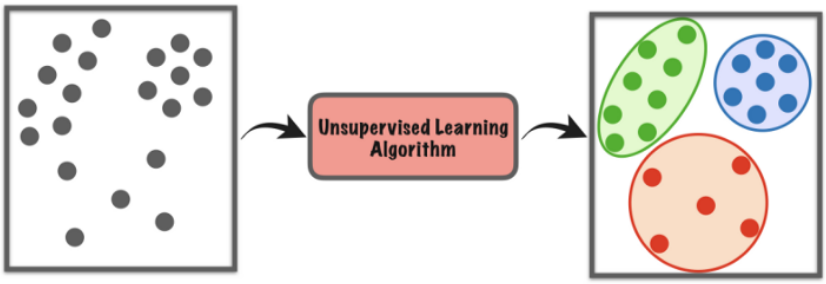

Figura 1.4: Apprendimento non supervisionato
Fonte: https://miro.medium.com/max/720/0*tamvSiqDneDfw2Vr

1.4 Deep Learning

Nonostante gran parte delle applicazioni di Machine Learning è basata sull'apprendimento supervisionato, la maggior parte dei dati disponibili non è etichettata. La fase di etichettatura richiede competenze, tempo e costi non indifferenti. È in questo contesto che assume importanza il Deep Learning. Con il Deep Learning i modelli possono apprendere anche da dati grezzi come testi o immagini e determinare automaticamente l'insieme di caratteristiche che distinguono una categoria di dati da un'altra. Vengono quindi eliminati gli interventi umani e viene ampliata la scelta di dati che si possono fornire al modello in fase di addestramento. Il cuore del Deep Learning sono le reti neurali.

1.5 Reti neurali

Così come i volatili spinsero l'uomo a volare e gli uncini dei fiori di bardana ispirarono l'ingegnere svizzero Georges de Mestral ad inventare il velcro, così l'architettura e l'organizzazione del cervello umano hanno dato ispirazione alla struttura delle reti neurali artificiali (ANN). Una rete neurale artificiale è un modello di Machine Learning la cui architettura è simile alla rete di neuroni che popola il nostro cervello.

Un neurone (figura 1.5) è composto dal corpo cellulare, o *soma*, contenente il nucleo e le componenti che si occupano delle principali funzioni cellulari, da alcuni prolungamenti chiamati *dendriti* e da uno più lungo chiamato *assone*. L'assone, in una delle sue estremità, si divide a sua volta in alcuni rami i cui estremi, le *sinapsi*, si collegano con i dendriti di altri neuroni. I neuroni producono piccoli impulsi elettrici che viaggiano attraverso gli assoni e fanno sì che le sinapsi rilascino dei segnali chimici, i *neurotrasmettitori*. Quando un neurone riceve una sufficiente quantità di neurotrasmettitori in pochi millisecondi, produce dei propri impulsi elettrici. [3]

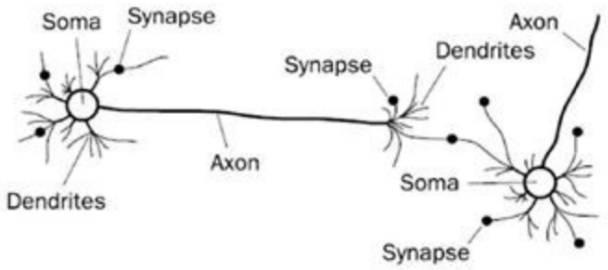

Figura 1.5: Struttura di un neurone biologico
Fonte: https://miro.medium.com/max/1400/1*ZjwXXxX3k50SvJsO7egYDQ.jpeg

Similmente al neurone biologico appena descritto si comporta il neurone artificiale, fondamentale nelle operazioni di una *ANN*. Come mostrato in figura (figura 1.6) ci sono tre elementi fondamentali in un neurone:

- Un insieme di collegamenti, ognuno dei quali è caratterizzato da un **peso**;
- Una **funzione lineare** che applica una somma pesata di tutti gli input ricevuti dal neurone;
- Una **funzione di attivazione** per limitare l'ampiezza dell'output del neurone.

In una rete neurale profonda (*Deep Neural Network*) troviamo, oltre allo strato di neuroni di input e allo strato di neuroni di output, uno o più strati intermedi nascosti (*hidden layers*). Aggiungere più strati nascosti permette alla rete di estrarre dall'input (come un testo o un'immagine) caratteristiche di alto livello per generare poi una rappresentazione gerarchica. Ogni strato aggiunge un livello di astrazione differente, che si occupa di un aspetto differente dell'input. La rete neurale nell'immagine (figura 1.7) è formata da *dense layers*, cioè ogni neurone è connesso a tutti quelli dello strato precedente. [4]

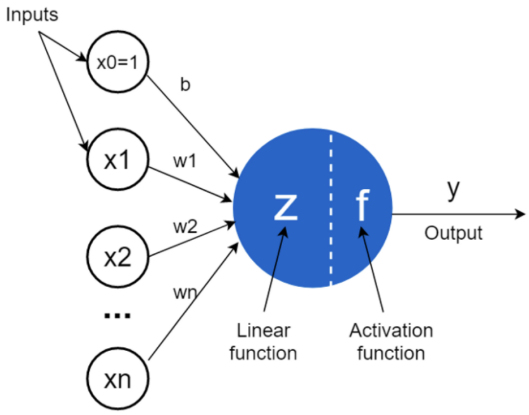

Figura 1.6: Struttura di un neurone artificiale
Fonte: https://miro.medium.com/max/1400/1*hkYlTODpjJgo32DoCOWN5w.png

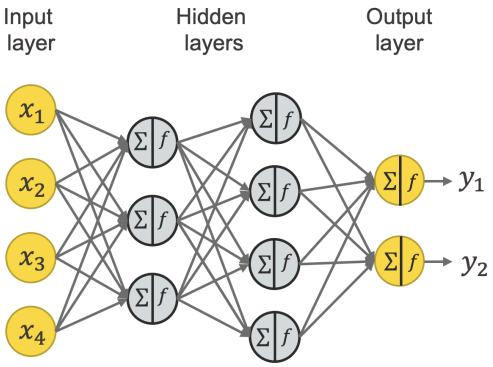

Figura 1.7: Deep Neural Network. A sinistra uno strato di quattro neuroni di input, al centro due strati nascosti, a destra lo strato di output formato da due neuroni
Fonte: https://miro.medium.com/max/1400/1*hkYlTODpjJgo32DoCOWN5w.png

Capitolo 2

Natural Language Processing

Con Natural Language Processing ci si riferisce a quella branca dell'Intelligenza Artificale che fornisce ai computer l'abilità di comprendere un testo scritto o parlato nello stesso modo in cui lo farebbe un essere umano. Il linguaggio umano è caratterizzato da molteplici amibiguità che rendono difficile la scrittura di software che determinino con precisione il significato di un testo o di un vocale. Omonimi, omofoni, ironia, metafore sono solo alcune irregolarità del linguaggio che richiedono anni per essere appresi da un essere umano. Un modello NLP però deve poter riconoscere ogni sfumatura del linguaggio per poter essere efficiente e soprattutto deve farlo in tempi molto più brevi. Tra i diversi task NLP:

- **Speech recognition** o **Speech-to-text**: convertire in modo efficiente i dati vocali in dati testuali. Il riconoscimento vocale è necessario per qualsiasi applicazione che si affidi a comandi vocali o che risponda a domande pronunciate a voce;

- **Machine Translation**: tradurre automaticamente un testo da una lingua a un'altra;

- **Sentiment analysis**: identificare lo stato d'animo e le opinioni soggettive all'interno di testi elaborati;

- **Information Retrieval**: processo di gestione di documenti di un archivio digitale in modo da facilitare all'utente, in seguito ad una sua ricerca, le informazioni che rispondono alla sua richiesta.

Un approccio comune per i task di NLP è utilizzare le *Reti Neurali Ricorrenti* o *RNN*.

2.1 RNN

Oltre alle reti neurali feed-forward dove le connessioni tra i neuroni sono soltanto in una direzione e quindi l'attivazione scorre dal layer di input verso il layer di output, esistono anche le reti neurali ricorrenti. La loro struttura è molto simile ma esistono anche connessioni che possono formare dei cicli. In figura 2.1 possiamo vedere il più semplice neurone RNN che riceve un input e produce un output che manda a se stesso. Ad ogni istante t (chiamato anche *frame*), questo neurone ricorrente riceve come input sia $\mathbf{x}_{(t)}$ che l'output prodotto nel frame precedente $\mathbf{y}_{(t-1)}$.

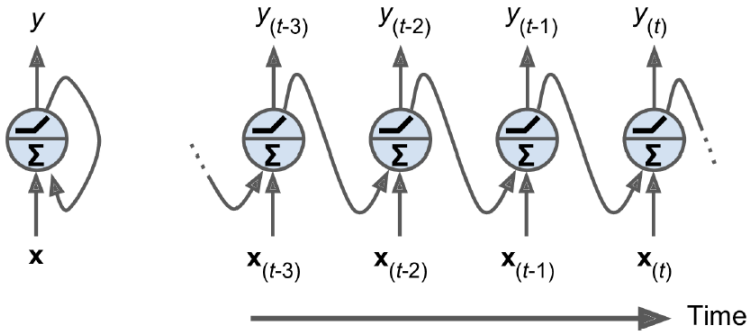

Figura 2.1: Un neurone ricorrente (a sinistra) e lo stesso neurone nel tempo (a destra)

Fonte: [3]

Dato che l'output di un neurone ricorrente al tempo t è una funzione di tutti gli input generati nei frame precedenti, possiamo dire che un neurone ha una sorta di memoria. Questa caratteristica sarà poi fondamentale per i task di NLP in quanto il significato di una parola in una frase non è a sé stante ma dipende dalle parole che la precedono e in generale dalle parole che la circondano.

Una RNN può essere di tipo *sequence-to-vector*. Un suo utilizzo lo troviamo nella Sentiment Analysis in cui da una sequence, come una recensione di un film, si ottiene un valore relativo al suo contenuto (-1 per un testo carico d'odio e +1 per uno carico di amore). Con un *vector-to-sequence* si può ottenere da un vettore, che rappresenta ad esempio un'immagine, una sequence di output come la descrizione di tale immagine. Infine si può avere sempre un *sequence-to-sequence* (figura 2.3) costuituito da un *sequence-to-vector*, chiamato **encoder** seguito da un *vector-to-sequence network*, chiamato **decoder**. Questa architettura prende il nome di **Encoder-Decoder** ed è molto utile nella traduzione di testi. Traducendo una frase da una lingua a un'altra un encoder produce un vettore

intermedio e un decoder, preso in input tale vettore, restituisce una sequenza che altri non è che la frase tradotta. Tale struttura è molto più efficiente che un semplice seq-to-seq network (figura 2.2) in quanto le ultime parole di una frase possono condizionare il significato delle prime, quindi è necessario ottenere un vettore che rappresenti l'intera frase prima di tradurla.

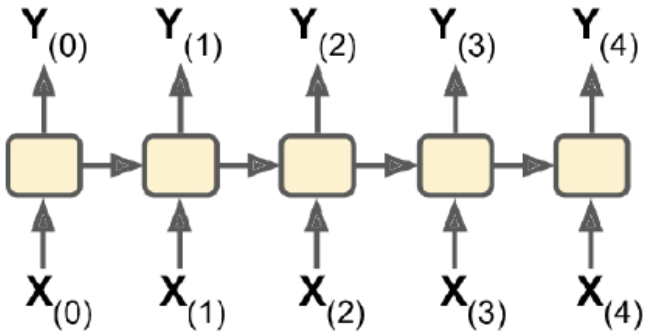

Figura 2.2: Sequence-to-sequence
Fonte: [3]

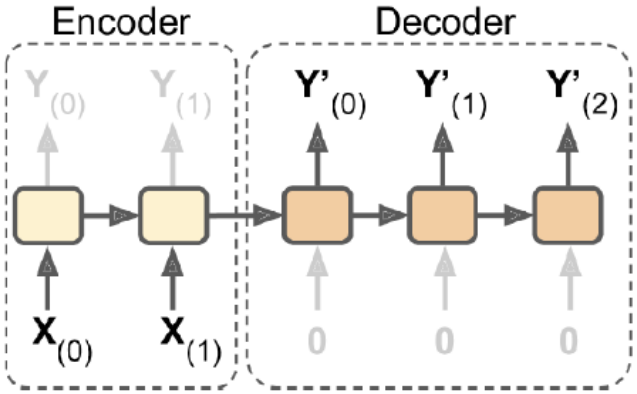

Figura 2.3: Encoder-Decoder
Fonte: [3]

2.2 Encoder-Decoder in un modello neurale di traduzione

In un semplice modello neurale di traduzione l'encoder processa ogni elemento nella sequenza di input, produce il *context*, un vettore di numeri reali che

contiene tutte le informazioni, e lo invia al decoder che produce una sequenza di output elemento per elemento (figura 2.4).

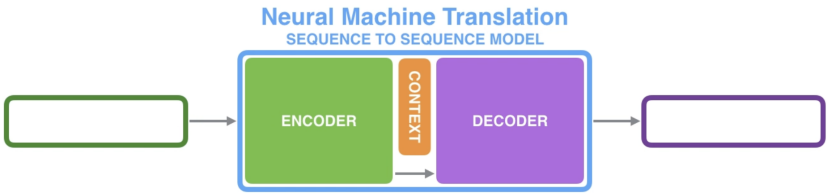

Figura 2.4: Architettura encoder-decoder e context
Fonte: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Una RNN prende due input per ogni frame: un input (nel caso dell'encoder una parola della frase) e un *hidden state*. La parola tuttavia deve essere rappresentata sottoforma di vettore. Per poterla trasformare si utilizzano degli algoritmi di *word embedding*.

2.2.1 Word embedding

Un word embedding è una conversione di testo dove parole che hanno significato simile hanno una rappresentazione simile. Questo approccio di rappresentare parole e documenti è considerato una delle scoperte chiave del deep learning sui problemi di elaborazione del linguaggio naturale. Ogni parola è trasformata in un vettore di valori reali di dimensione nell'ordine delle decine o centinaia. Questo algoritmo è un'importante innovazione rispetto alle rappresentazioni sparse come il *One-hot encoding* in cui ogni parola ha una rappresentazione univoca rispetto al testo cui appartiene o addirittura, nel caso del task di traduzione, rispetto a un dizionario e che quindi richiede vettori di dimensione nell'ordine delle migliaia (o addirittura dei milioni).

Ritornando al nostro modello neurale di traduzione si possono utilizzare algoritmi di embedding pre-addestrati o anche addestrare un proprio embedding sul proprio dataset.

2.3 Attention

Il vettore *context* si è rivelato essere un collo di bottiglia per questi tipi di modelli. Osserviamo per esempio la figura 2.5. Consideriamo il percorso dalla parola *milk* alla sua traduzione *lait*. Possiamo osservare che la rappresentazione di questa parola necessita di essere trasportata per diversi step prima di essere effettivamente utilizzata. La difficoltà maggiore infatti si presenta quando bisogna elaborare delle frasi lunghe. L'hidden state di una parola percorre un

tragitto molto lungo con il forte rischio di perdere informazioni. Per risolvere questi problemi caratteristici dei LSTM (long-short term memory network) si utilizza la tecnica dell'*Attention* che ha rivoluzionato i modelli neurali di traduzione (e il NLP in generale) migliorando notevolmente la qualità dei sistemi di traduzione automatica. Attention permette al modello di concentrarsi sulle parti rilevanti della frase di input. [5]

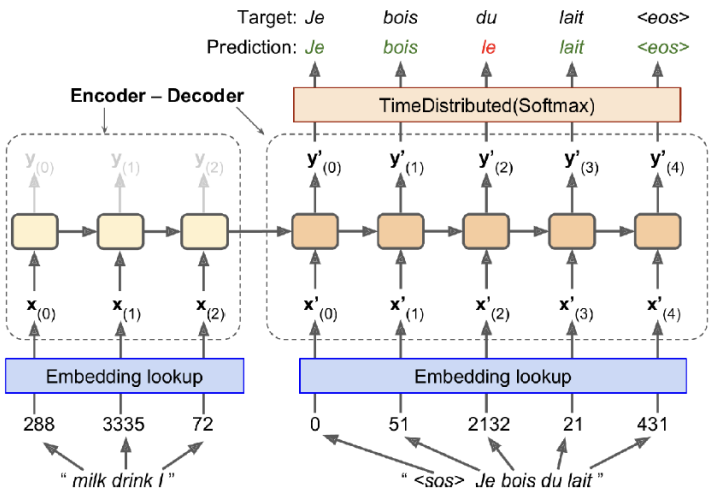

Figura 2.5: Un semplice modello encoder-decoder di traduzione
Fonte: [3]

Un attention model differisce da un classico modello seq-to-seq in quanto l'encoder trasmette al decoder non solo l'ultimo hidden state ma tutti quanti. Inoltre, il decoder, per ogni hidden state ricevuto, assegna un punteggio che viene moltiplicato per un altro valore (*softmaxed score*) che ha lo scopo di amplificare gli hidden state con punteggio alto, e limitare gli hidden state con punteggio basso. Così facendo, ad ogni step, il decoder è consapevole della parola su cui focalizzarsi. [6]

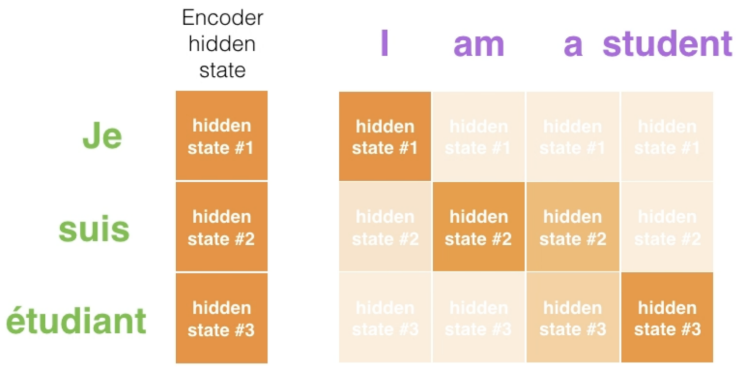

Figura 2.6: Esempio di utilizzo degli hidden state con attention
Fonte: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Una caratteristica fondamentale dell'Attention è l'*Explainability*. Questa feature, molto importante nel mondo dell'Intelligenza Artificiale, è molto utile per capire come migliorare un modello in caso di errori. Prendiamo ad esempio un'immagine di "un cane sulla neve" per la quale il modello produce una descrizione "lupo sulla neve". È possibile trovare su cosa si è posta l'attenzione del modello nel momento in cui produceva l'output "lupo". Probabilmente non solo sull'animale ma anche sulla neve, poiché in fase di addestramento aveva notato che la differenza tra lupi e cani fosse la presenza o meno della neve. Questo errore potrebbe essere corretto in fase di training con delle foto di lupi senza neve e di cani con la neve. [7] In molte applicazioni l'explainability non è solo un modo di debugging ma un requisito legale (come in un sistema che deve decidere se concedere o no un mutuo).

La fase di addestramento di una RNN con Attention è però molto lunga poiché l'embedding di una frase avviene trasformando una parola dopo l'altra. È qui che entra in gioco l'innovativa architettura del Transformer.

2.4 Transformer

Nel 2017 un team di ricercatori Google ha scritto un saggio sorprendente chiamato "Attention Is All You Need". [8] L'innovazione sta nell'utilizzare l'Attention per migliorare la velocità in fase di traning grazie alla parallelizzazione. Questa nuova architettura, il *Transformer*, ha migliorato lo stato dell'arte nell'NMT (Neural Machine Translation) senza usare RNN o CNN (Convolutional Neural Network).

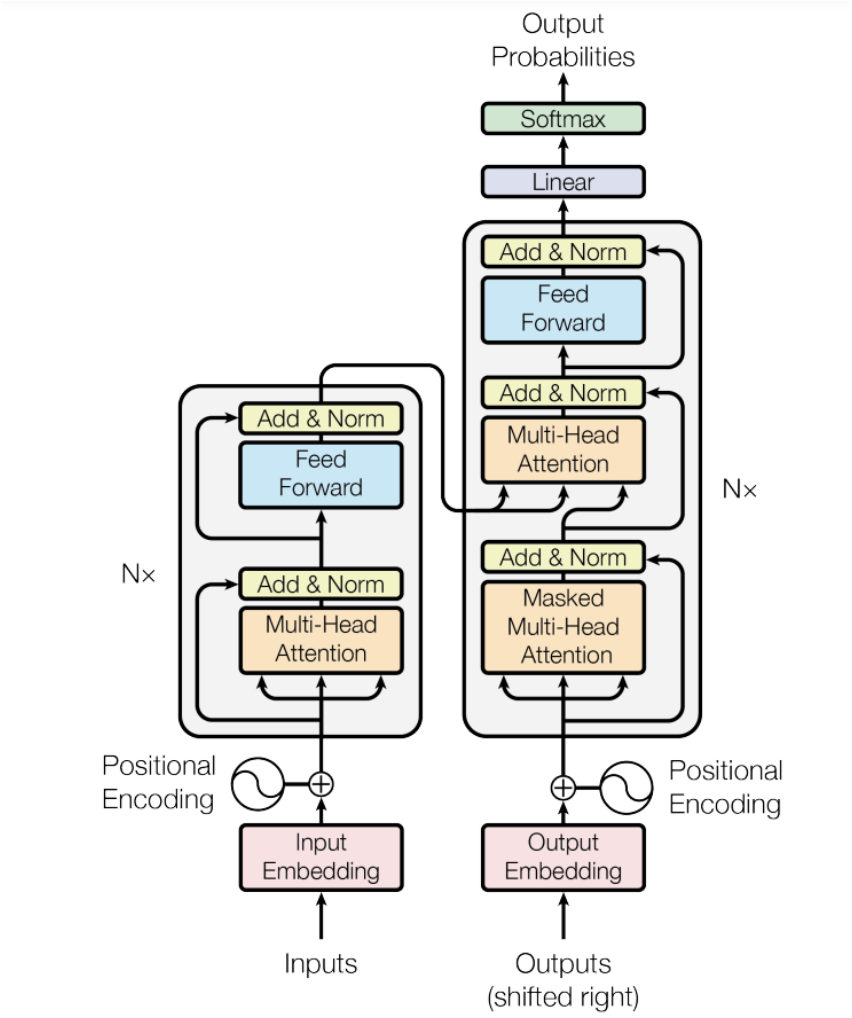

Figura 2.7: Architettura del Transformer
Fonte: [8]

In figura 2.7 vediamo l'architettura di un Transformer. Prendiamo il caso della traduzione dall'inglese di "The big red dog" al francese "Le gros chien rouge". Sulla sinistra l'encoder prende in input la frase rappresentata da una sequenza di parole. L'embedding di una parola, in un Transformer, contiene anche una codifica della sua posizione rispetto alla frase (una parola può assumere un significato diverso a seconda di dove è posizionata). L'embedding è trasmesso al Multi-Head Attention che ridefinisce il ruolo dell'Attention migliorando l'abilità del modello nel focalizzarsi su più parole della frase contemporanemante. In figura 2.8 si può osservare come si comporta un Multi-Head Attention (solitamente composto da 8 teste, da qui il nome Multi-Head) durante l'encoding della parola *it* sulla destra. Una *Attention Head*

(in arancione) pone l'attenzione maggiore su *animal*, un'altra (in verde) pone l'attenzione maggiore su *tired*. In un certo senso la rappresentazione di *it* contiene informazioni sia su *animal* che su *tired* che difatti sono rispettivamente il soggetto cui si riferisce il pronome e un suo aggettivo. Il vettore di parola è una media pesata della concatenazione degli 8 vettori restituiti da ogni Attention Head. Un vettore ora ha quindi informazioni relative a diverse parole a cui molto probabilmente è relazionata (come un soggetto di una frase è in relazione ad un articolo, ad un verbo, ad un aggettivo). Ogni vettore passa poi per un feed-forward network il cui scopo principale è di ottenere un risultato di dimensione opportuna per i successivi step. L'innovazione sta nel fatto che ogni Attention Block è indipendente dagli altri perciò questo processo è parallelizzabile.

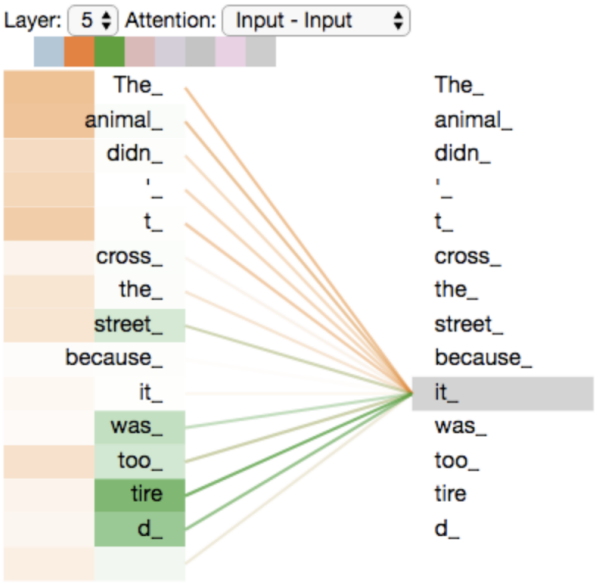

Figura 2.8: Esempio grafico di Multi-Head Attention
Fonte: https://jalammar.github.io/illustrated-transformer/

Il decoder ha una struttura simile all'encoder. Durante il processo di traduzione viene dato in input al Multi-Head Attention sia l'output dell'intera frase dell'encoder, sia l'embedding della frase nella seconda lingua ("Le gros chien rouge"). L'embedding però nasconde tutte le parole successive a quella che al momento si sta elaborando (da qui il nome *Masked Multi-Head Attention*). Se quindi si sta processando la parola "chien" gli input del decoder sono l'output dell'encoder e l'embedding di "Le gros". Tramite il Multi-Head Attention si costruisce una sorta di mappa che pone in relazione ogni vettore (ogni parola) dell'encoder (della prima lingua) con ogni vettore della frase nella seconda

lingua. Il decoder infine, con un softmax layer, produce in output la parola che più probabilmente segue quelle già tradotte.

2.5 BERT

BERT (Bidirectional Encoder Representations from Transformers) è un modello di Deep Lerning pre-addestrato dai ricercatori di Google AI Language. [9] Il Transformer può essere diviso fisicamente in encoder e decoder separando i suoi task. L'encoder infatti impara di una lingua la sua ortografia, la sua morfologia e la sua sintassi, in poche parole la sua grammatica. Il decoder impara a mappare le parole di una lingua in quelle di un'altra. BERT utilizza solo l'architettura dell'encoder per risolvere alcuni task che richiedono la comprensione della lingua:

- Neural Machine Translation
- Question Answering
- Sentiment Analysis
- Text summarization

Quindi si può utilizzare un modello Bert addestrato nella comprensione di una lingua e realizzare un fine-tuning in base al problema da risolvere. Per ottenere ciò i passaggi chiave sono:

- Pre-training: comprendere cosa è una lingua e qual è il contesto grazie all'addestramento simultaneo su due task non-supervisionati:
  - Masked Language Model (MDM): viene presa una frase in input con alcuni token (parole) casuali nascosti. L'obiettivo è predirre le parole mascherate.
  - Next Sentence Prediction (NSP): da due frasi in input, Bert determina se la seconda frase è successiva alla prima. Questo task aiuta Bert a capire il contesto attraverso diverse frasi.
- Fine-tuning: si può addestrare Bert in un task specifico. Per esempio in un task di Question-Answering vengono usati come input una domanda e il paragrafo dove ricercare la risposta. Come output si ottiene l'indice dell'inizio e della fine della frase che racchiude la risposta.

Addestrandosi su questi due task Bert simula l'apprendimento di una lingua da parte di un essere umano: predirre le parole e capire il contesto che lega più frasi.

2.6 SentenceBert

Bert e RoBERTa (un estensione di Bert) hanno raggiunto lo stato dell'arte nel campo dei task di *sentence-pair regression* come la *Semantic Textual Similaruty* (STS). Tuttavia questo task richiede di elaborare entrambe le frasi e ciò impone un enorme sovraccarico di calcolo. Cercare le frasi semanticamente più simili in una collezione di 10.000 frasi richiede circa 50 milioni di confronti ($\sim$ 65 ore). Con Sentence-Bert (SBERT), che è un'estensione di BERT che utilizza *siamese network* e *triplet network*, l'operazione di ricerca delle due frasi più simili richiede circa 5 secondi, mantenendo la stessa accuratezza di BERT. [10] SBERT è un framework di Python per l'embedding allo stato dell'arte di frasi, testo e immagini.

2.7 Semantic Textual Similarity

Il task di Semantic Textual Similarity consiste nel confrontare due frasi (o addirittura due testi) e affermare se esse abbiano un significato simile. Il STS è un problema notoriamente complesso a causa delle sfumature del linguaggio naturale. Due testi possono essere semanticamente simili pur non avendo alcuna parola in comune. L'idea chiave di SBERT si configura nei seguenti passaggi:

- Utilizzare SNLI dataset[1] o STS dataset[2] come dati di training. Questi dataset contengono diverse migliaia di coppie di frasi etichettate come simili o dissimili;

- Per ogni testo del dataset di training viene realizzato un embedding con l'encoder di un BERT pre-addestrato;

- Viene eseguita la *Mean Pooling* per ogni token del testo ottenendo così un embedding di lunghezza fissa;

- Il modello viene addestrato usando un'architettura *Siamese Network* con Contrastive Loss. In tal modo gli embedding di testi simili risulteranno essere più vicini nell'*Embedding Space* mentre gli embedding di testi dissimili verranno vettorialmente allontanati;

- Dopo la fase di training è possibile comparare qualsiasi coppia di testi calcolando la similarità coseno tra gli embedding dei due testi.

[1] `https://paperswithcode.com/dataset/snli`
[2] `https://paperswithcode.com/dataset/semantic-textual-similarity-2012-2016`

La similarità coseno (2.1) è una tecnica per la misurazione del grado di similarità tra due vettori effettuata calcolando il coseno fra essi.

$$\cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{ab}}{\|\mathbf{a}\|\|\mathbf{b}\|} = \frac{\sum_{i=1}^{n} \mathbf{a}_i \mathbf{b}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{a}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{b}_i)^2}} \tag{2.1}$$

2.8 Chatbot

I chatbot aiutano le organizzazioni a risolvere le domande dei clienti, a perferzionare la comunicazione e a migliorare l'esperienza dei consumatori. I chatbot traducono il linguaggio umano in informazioni digitali applicando tecniche di machine learning e di NLP. Esistono diversi tipi di chatbot a seconda del modo di processamento degli input e della generazione delle risposte. Tra questi, il *Generative chatbot* e il *Retrieval-Based chatbot*.

2.8.1 Retrieval-based chatbot

I Retrieval-based chatbot sono addestrati nel fornire la migliore risposta possibile da un database di risposte predefinite. Questa tipologia di chatbot usa tecniche come *keywords matching*, *machine learning* o *deep learning* per identificare la risposta migliore. Indipendentemente dalla tecnica, questi chatbot forniscono solo risposte predefinite e non generano nuovo output.

2.8.2 Generative chatbot

I Generative chatbot possono generare nuovi dialoghi basandosi su una grande quantità di dati di training. Questa tipologia di chatbot utilizza tecniche combinate di apprendimento supervisionato, non supervisionato e per rinforzo.

L'apprendimento supervisionato, oltre ad avere difficoltà nell'includere nomi propri per via della scarsa presenza nei dataset, risultano ripetitivi e non riescono ad instaurare una realistica conversazione. Per affrontare questo problema, gli sviluppatori utilizzano l'apprendimento per rinforzo per insegnare ai chatbot come ottimizzare i dialoghi ed ottenere delle ricompense.

Capitolo 3

Progetto

3.1 Introduzione

Obiettivo del progetto è creare un HelpDesk per un'azienda che fornisca assistenza e supporto informativo ad un utente. Nello specifico un utente pone una domanda ad un chatbot ed ottiene una risposta predefinita che possa aiutarlo nel risolvere il proprio problema. Un chatbot è un software che simula e processa una conversazione con una persona. I chatbot possono essere semplici e fornire una singola risposta ad una domanda (*query*), oppure più complessi. Nel secondo caso il loro comportamento è simile a quello di un assistente digitale che quindi impara e si evolve durante la conversazione per fornire livelli crescenti di personalizzazione mentre raccoglie ed elabora le informazioni. L'azienda in questione ha fornito una collezione di circa 50 domande in italiano con le annesse risposte. Il software, una volta ottenuta in input la domanda dell'utente, deve cercare la domanda relativa nella collezione e restituire la rispettiva risposta. L'utente però può sottoporre al chatbot una domanda non esattamente identica ma semanticamente simile ad una delle domande della collezione. In tal caso il software, essendo un task di NLP, deve riconoscere la domanda semanticamente più simile proprio come se fosse un essere umano.

3.2 Formulazione della soluzione

Per poter implementare un modello capace di risolvere questo task si è pensato di utilizzare SBERT. Come riporta la documentazione ufficiale di SBERT[1], può essere utile per i task di *semantic textual similarity* (STS), *semantic search*

[1] `https://www.sbert.net/index.html`

(SS) e *paraphrase mining* (PM). È un framework basato su Pytorch e Transformers e offre una vasta collezione di modelli pre-addestrati specializzati su svariati task. Il task di questo progetto viene risolto tramite *Semantic Textual Similarity* (STS), poiché SS e PM sono più adatti per affrontare ricerche semantiche in grandi collezioni di frasi (oltre i 10.000 esempi).

3.3 Scelta del modello

Il primo problema che si pone davanti è che tra i modelli pre-addestrati di SBERT ufficiali non esiste ad oggi un modello per la sola lingua italiana. Esistono tuttavia dei modelli multi-lingua addestrati su più lingue diverse tra cui l'italiano. Inoltre, sono stati trovati altri modelli sulla community di Hugging Face[2] che corrispondevano al task di STS per la lingua italiana. In tutto sono stati selezionati 7 modelli:

- Da SBERT.net:
  - distiluse-base-multilingual-cased-v1 [11]
  - distiluse-base-multilingual-cased-v2 [11]
  - paraphrase-multilingual-mpnet-base-v2 [11]
  - paraphrase-multilingual-MiniLM-L12-v2 [11]
- Da Hugging Face:
  - clips/mfaq [12]
  - aiknowyou/aiky-sentence-bertino
  - efederici/sentence-bert-base [13]

3.4 Scelta del dataset per la sperimentazione

Poichè il caso di studio è reale, la struttura e le dimensioni ridotte del dataset non ci permettono né di valutare accuratamente i modelli né di fare un fine-tuning sul dominio. Si è scelto perciò il dataset *stsb_multi_mt*[3] [14] che contiene un sottoinsieme in lingua italiana. In questo dataset sono presenti traduzioni multilingua dal dataset originale inglese di *STSbenchmark*. La traduzione è stata fatta con DeepL[4]. STSbenchmark comprende una selezione dei

[2] `https://huggingface.co/`

[3] `https://huggingface.co/datasets/stsb_multi_mt`

[4] `https://deepl.com`

dataset in inglese utilizzati nei task STS organizzati nei workshop internazionali di SemEval tra il 2012 e il 2017. Le frasi selezionate includono didascalie di immagini, titoli di notizie e forum di utenti. In figura 3.1 vengono presentate alcune frasi per mostrare la struttura del dataset. Sono presenti 8.628 elementi. Ogni elemento è composto da una coppia di frasi e dalla rispettiva similarità coseno. La similarità coseno in questo caso va da 0 a 5 ed esprime numericamente quanto sono semanticamente simili due frasi. In verde è mostrato un esempio di perfetta similarità, in rosso un esempio di totale assenza di similarità, in blu un esempio di parziale similarità. Il dataset è composto da:

- *train set*: 5749 elementi
- *validation set*: 1500 elementi
- *test set*: 1379 elementi

sentence1 (string)	sentence2 (string)	similarity_score (float32)
"Una donna sta affettando del tonno."	"Una donna sta tagliando il pesce crudo."	2.8
"Un uomo che suona la chitarra."	"Una donna si dipinge le labbra."	0
"Un uomo sta suonando un flauto."	"Un cane abbaia ad una mosca."	0
"Un piccolo rinoceronte cammina intorno alla sua penna con la madre."	"Un rinoceronte bambino sta seguendo un rinoceronte adulto."	3.4
"L'uomo ha usato una spada per tagliare una bottiglia di plastica."	"Un uomo ha affettato una bottiglia di plastica con una spada."	5
"Un uomo sta pulendo un pesce sul bancone di una cucina."	"Un uomo sta suonando un flauto."	0.8
"Una persona sta tagliando le foglie di coriandolo."	"Una donna sta tagliando delle foglie verdi."	2.25
"Un uomo sta premendo dei pulsanti a microonde."	"Un uomo accende il microonde."	2.75
"Una ragazza è a cavallo."	"La ragazza ha trotterellato il cavallo."	4.5
"Batman e Robin fanno volare un elicottero sull'acqua."	"Un elicottero vola sull'acqua."	2.6
"Una donna versa olio in una padella da una bottiglia di plastica mentre parla."	"Una donna anziana sta versando l'olio in una padella sulla stufa."	3.8

Figura 3.1: Alcuni esempi del dataset *stsb_multi_mt*

In tabella 3.1 è stata verificata e confermata l'equa distribuzione delle classi dei tre split del dataset.

Classe	**Training Set**	**Validation Set**	**Test Set**
0-1:	1103	389	308
1-2:	1076	303	291
2-3:	1297	359	352
3-4:	1942	421	442
4-5:	1406	264	338

Tabella 3.1: Distribuzioni delle classi dentro il dataset.

3.5 Metriche di valutazione

Per valutare il modello si è calcolata la similarità coseno di ogni coppia di frasi del *test set* e si è successivamente misurata l'accuratezza con *MSE*, *MAE*, *Exact Match* e *Rank-n*.

Per calcolare la similarità coseno si è ottenuto l'embedding di tutte le prime delle due frasi di ogni coppia (sentence1_set), l'embedding delle seconde (sentence2_set) e si è creata una matrice contenente al posto (i,j) la similarità coseno tra la frase i del primo set e la frase j del secondo set. Poi sono state opportunamente selezionate le coppie di frasi corrette.

```
#Embedding di sentence1_set
embeddings_sentence1_set = model.encode(sentence1_set,
    convert_to_tensor=True)

#Embedding di sentence2_set
embeddings_sentence2_set = model.encode(sentence2_set,
    convert_to_tensor=True)

#Calcolo della similarita coseno
cosine_scores = util.cos_sim(embeddings_sentence1_set,
    embeddings_sentence2_set)
```

3.5.1 MAE, MSE, Exact Match

Per verificare che la similarità ottenuta fosse corretta si è prima calcolato il *Mean Absolute Error* (MAE). Il MAE consiste nel calcolare l'errore assoluto medio. Con errore si definisce la differenza tra la similarità coseno reale, e quindi già presente nel dataset, e quella previsto dal modello. Questa misura dà un'indicazione dello scostamento medio.

$$MAE = \frac{\sum_{i=1}^{n} |RealScore_i - PredictedScore_i|}{n} \tag{3.1}$$

Successivamente si è calcolato il *Mean Squared Error* (MSE). Il MSE consiste nel calcolare lo scarto quadratico medio. A differenza del MAE, gli errori più grandi hanno un impatto maggiore rispetto a quelli piccoli.

$$MSE = \frac{sum_{i=1}^{n}(x_i - y_i)^2}{n} \tag{3.2}$$

Infine, è stato preso in considerazione l'*Exact Match* (EM). Una volta arrotondati gli score all'intero più vicino, si è contato quante volte le predizioni

fossero uguali ai valori originali. Si sono usate queste tre metriche di valutazione per ognuno dei modelli e i risultati ottenuti sono i seguenti:

Modello	**MAE**	**MSE**	**EM**
distiluse-base-multilingual-cased-v1	**0.85**	**1.16**	<u>35.68%</u>
distiluse-base-multilingual-cased-v2	0.93	1.40	34.40%
paraphrase-multilingual-mpnet-base-v2	0.93	1.41	34.70%
paraphrase-multilingual-MiniLM-L12-v2	**0.89**	<u>1.34</u>	**37.35%**
clips/mfaq	2.30	7.49	11.24%
aiknowyou/aiky-sentence-bertino	1.87	5.15	16.24%
efederici/sentence-bert-base	1.05	1.78	31.70%

Tabella 3.2: Calcolo delle accuratezze dei modelli. I risultati migliori sono evidenziati in grassetto, i secondi sono sottolineati.

Come visibile dalla tabella 3.2, i modelli migliori sono quelli di SBERT. Il modello con i valori più incoraggianti è *distiluse-base-multilingual-cased-v1*.

3.5.2 Rank-n

Per il nostro obiettivo il criterio di valutazione più interessante è il *Rank-n*. Per poter ottenere questa metrica si è modificata la struttura del training set in modo da ottenere un dataset di sole coppie di frasi simili. È stato stabilito che due frasi sono simili se il coseno di similarità tra di esse è maggiore di 4 (su 5). Essendo un criterio che verrà usato per tutti i modelli, cambiando il valore, ad esempio in 4.3, si ottengono valori diversi ma l'ordine rimane invariato. Se quindi con 4 il modello A ha risultati migliori del modello B, anche scegliendo 4.3 A sarà migliore di B. Per ogni modello si è calcolato *Rank-1*, *Rank-2* e *Rank-3*. Per calcolare il *Rank-n* si è preso `sentence1_set` e per ognuna delle 231 frasi si è cercata la rispettiva "gemella" all'interno di `sentence2_set`. Per ogni ricerca viene creata una lista di frasi in ordine decrescente di score. La prima posizione è quindi occupata dalla frase che per il modello è più simile alla frase in questione. Al secondo posto quella che ha il secondo score di similarità più alto, e così via. Il Rank-1 conta quante volte il modello trova la gemella. Il Rank-2 conta quante volte il modello trova la gemella nelle prime due posizioni. Il Rank-3 conta quante volte il modello trova la gemella nelle prime tre posizioni. Per un'analisi più approfondita, sono state testate diverse combinazioni dei due `sentence_set`. A seguire la media dei valori ottenuti:

- Rank-1: 82%

- Rank-2: 90%
- Rank-3: 94%

3.6 Fine-tuning

Con il termine *fine-tuning* si intende l'addestramento di un modello che è stato già addestrato per un task simile. Questa scelta ha un doppio possibile risvolto positivo. Il primo quando il dataset in possesso non è sufficientemente grande per addestrare un modello "from scratch", il secondo è che l'addestramento viene fatto usando in partenza i pesi del modello pre-addestrato e, modificandoli, il modello viene specializzato sul proprio dataset. Non è detto che ci sia sempre un miglioramento quindi è stato valutato il modello anche dopo il fine-tuning con il rank-n. Nello specifico il fine-tuning è stato fatto sul training set italiano di *stsb_multi_mt* usando la CosineSimilarityLoss. Con questa funzione, per ogni coppia di frasi, vengono creati gli embedding delle due frasi. Successivamente viene calcolata la similarità coseno tra gli embedding e il risultato viene comparato con il reale score (gold label).

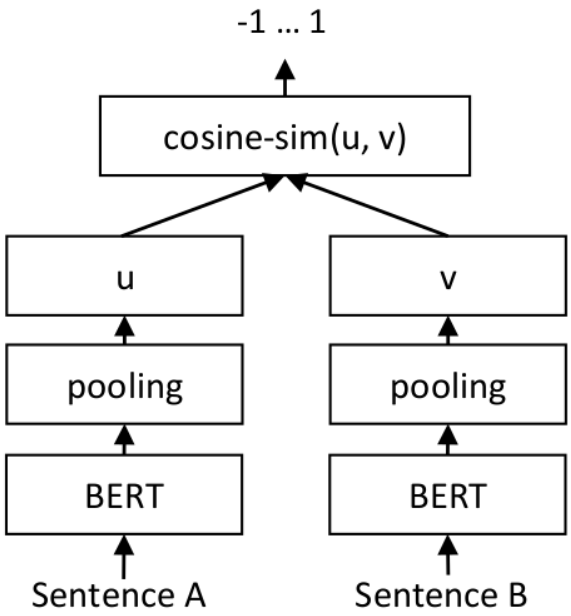

Figura 3.2: CosineSimilarityLoss con range da -1 a 1
Fonte: https://www.sbert.net/docs/package_reference/losses.html#cosinesimilarityloss

3.6.1 Valutazione dopo il fine-tuning

Dopo il fine-tuning è stata misurata l'accuratezza del modello per poterlo confrontare con quello precedente. In tabella 3.3 i valori di MAE, MSE ed EM.

Modello	**MAE**	**MSE**	**EM**
distiluse-base-multilingual-cased-v1 (fine-tuned)	0.66	0.77	45.03%

Tabella 3.3: Calcolo delle accuratezze del modello dopo il fine-tuning.

A seguire la media dei valori ottenuti:

- Rank-1: 82%
- Rank-2: 90%
- Rank-3: 95%

È migliorato soltanto il Rank-3 di 1 punto percentuale tuttavia c'è stato un notevole miglioramento nell'MAE, nell'MSE e nell'ME. In particolare il valore di MSE indica che il modello è più preciso rispetto a prima nel calcolare la similarità coseno. Questo sarà utile nei successivi esperimenti in quanto sarà importante la differenza di score tra la frase più simile e la seconda classificata.

3.7 Tempi di esecuzione

Perchè sia un software realmente utilizzabile è necessario che i tempi di elaborazione siano minimi. È stato simulata l'immissione di una domanda da parte di un utente e la successiva elaborazione all'interno di un Colab.

- CPU: Intel(R) Xeon(R) CPU @ 2.00GHz
- GPU: Tesla T4

Per elaborare l'embedding delle 50 domande del dataset proprietario sono stati necessari 0.31 secondi. Questo embedding è però eseguito offline ed è già pronto all'uso al momento dell'inserimento della domanda. Per elaborare la frase più simile e sottoporre la risposta ci vogliono 0.01 secondi.

3.8 Metric Learning

Analizzando i dati, il metric learning crea una nuova tipologia di distanza. Applicare il metric learning permette di distinguere meglio i dati. Lo scopo principale è di creare una nuova metrica che minimizzi le distanze tra dati della stessa classe e che massimizzi le distanze tra dati di classi differenti. In pratica si cerca di raggruppare le istanze con caratteristiche simili. Riconoscimento facciale, Semantic Textual Similarity, verificazione vocale, similarità tra pazienti sono tutti task basati sulla similarità. Esistono diverse tecniche di metric learning. A seconda della loss function utilizzata, le istanze vengono avvicinate o allontante in modo differente.

Tra le loss function più comuni ci sono la *Triplet Loss* e la *Contrastive Loss*.

3.8.1 Triplet Loss

La *Triplet Loss* è probabilmente la loss function più popolare. Per poter utilizzare questa funzione occorre una collezione di triplette di frasi nella forma seguente.

$$(x_{ia}, x_{ip}, x_{in}) \tag{3.3}$$

In ogni tripletta i: *anchor* è una frase, *positive* è una frase semanticamente simile e *negative* è una frase semanticamente dissimile. Lo scopo della Triplet Loss è fare sì che la distanza

$$D_{ia,ip} = ||f(x_{ia}) - f(x_{ip})||_2 \tag{3.4}$$

sia minore della distanza

$$D_{ia,in} = ||f(x_{ia}) - f(x_{in})||_2 \tag{3.5}$$

di almeno un fattore α.

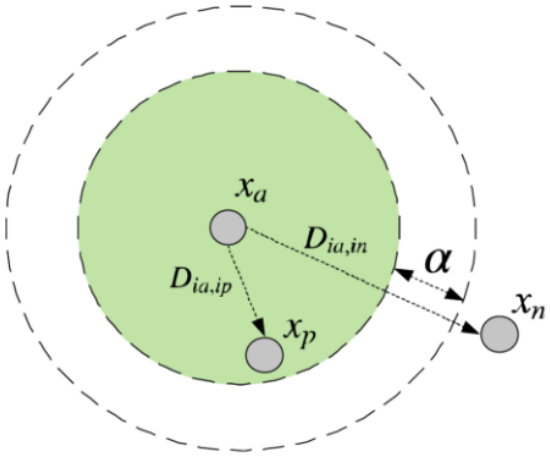

Figura 3.3: Triplet Loss
Fonte: https://towardsdatascience.com/metric-learning-loss-functions-5b67b3da99a5

Il modello è stato quindi addestrato sul training set con questa nuova struttura. Sono state provate due diverse strutture. Per scegliere la coppia di frasi positiva sono state selezionate tutte le coppie di frasi con score maggiore di 4. Per quanto riguarda la negativa nel primo caso è stata scelta una frase randomicamente tra tutte le frasi con score minore o uguale a 4, nel secondo caso è stata scelta la migliore negativa, ossia quella che il modello ha ritenuto essere la frase più simile all'anchor esclusa la positiva. In tal modo il modello probabilmente impara a distinguere con più precisione frasi simili ma non abbastanza.

Modello	**Rank-1**	**Rank-2**	**Rank-3**
distiluse-base-multilingual-cased-v1 con negativa casuale	81%	90%	93%
distiluse-base-multilingual-cased-v1 (fine-tuned) con negativa casuale	82%	90%	94%
distiluse-base-multilingual-cased-v1 con best negative	79%	87%	91%
distiluse-base-multilingual-cased-v1 (fine-tuned) con best negative	**83%**	**90%**	**94%**

Tabella 3.4: Calcolo delle accuratezze dei modelli. I risultati migliori sono evidenziati in grassetto.

3.8.2 Contrastive Loss

Similmente alla triplet loss, la contrastive loss cerca di avvicinare le frasi simili e di allontanare quelle dissimili. Per fare ciò occorre una collezione di

triplette di elementi nella forma seguente.

$$(x_{i1}, x_{i2}, l) \tag{3.6}$$

In ogni tripletta i: x_{i1} è una frase, x_{i2} un'altra frase e l ha valore 1 se le frasi sono simili, 0 se le frasi sono dissimili. Con la Contrastive Loss (3.7) il modello elabora l'embedding delle due frasi e la loro similarità. Se due frasi vengono ritenute simili ma la loro label è 0 allora vengono distanziate di almeno un fattore α, se le due frasi vengono ritenute dissimili ma la loro label è 1 allora vengono avvicinate. Lo scopo della Contrastive Loss è migliorare l'embedding del modello facendo sì che frasi simili siano più vicine nell'embedding space e frasi dissimili siano più distanti.

$$L_{contrastive} = l * ||x_{i1} - x_{i2}|| + (1 - l) * max(\alpha - ||x_{i1} - x_{i2}||; 0) \tag{3.7}$$

I risultati ottenuti sono i seguenti:

Modello	**Rank-1**	**Rank-2**	**Rank-3**
distiluse-base-multilingual-cased-v1 (fine-tuned) con contrastive loss	80%	88%	92%

Tabella 3.5: Calcolo della accuratezza del modello

3.9 Applicazione sul dataset proprietario

Dopo aver ottenuto un miglioramento nei Rank-n con un modello *distiluse-base-multilingual-cased-v1* fine-tuned sul dataset italiano *stsb_multi_mt* e Triplet Loss (con *best negative*), è stato applicato il modello al caso di studio reale. Il dataset aziendale è popolato da circa 50 coppie domanda-risposta perciò per poter testare l'effettiva qualità del modello sono necessarie molte più frasi semanticamente simili a quelle del dataset in modo da poter simulare l'immissione in input di una domanda da parte di un utente. A tal punto si è dovuto fare un processo di *data augmentation* per avere un quantitativo di dati sufficiente e avere dei risultati precisi.

3.9.1 Data-augmentation

Con un processo di *translation* e *back-translation* sono state generate 10 frasi sinonime per ognuna delle domande. Grazie a No Language Left Behind (NLLB) ogni domanda è stata tradotta in 10 lingue differenti. Successivamente ogni risultato ottenuto è stato ritradotto in italiano. Così facendo si ottengono,

quasi sempre, frasi diverse ma semanticamente simili. NLLB è un progetto di AI, primo nel suo genere, capace di creare modelli di traduzioni altamente accurati tra più di 200 lingue, comprese lingue meno conosciute come l'asturiano, il luganda, l'urdu e molte altre. [15]

Le lingue utilizzate sono state: spagnolo, inglese, tedesco, francese, olandese, hindi, portoghese, rumeno, russo e arabo. La similarità coseno tra ogni traduzione e la relativa frase/domanda di partenza è di 4.6 confermando la validità del modello di traduzione.

3.9.2 Test

Alcune traduzioni risultano identiche a quella originale perciò è stato fatto un processo di pulizia dei dati per non alterare il test. Prima della fase di testing viene preparato l'embedding di tutte le domande presenti nel dataset aumentato. Sia I il dataset proprietario con n domande e sia A il dataset aumentato. Per valutare il modello si è scelta una domanda casuale D_i tra i sinonimi d_i di una domanda $i \in I$. Successivamente si è calcolata la similarità coseno tra la domanda scelta D_i e tutte le domande del nostro dataset aumentato A escludendo la domanda scelta. Tra tutti gli score di domande simili è stato scelto lo score maggiore in modo da ottenere un vettore contenente gli n score migliori per ogni domanda i. Ordinando in modo decrescente sono state ottenute le 3 domande più simili alla domanda posta dall'utente fittizio. Questo processo è stato fatto per ogni i e quindi per n volte. Il test è stato ripetuto per 4 volte e il Rank-n medio ottenuto è il seguente:

- Rank-1: 95%
- Rank-2: 99%
- Rank-3: 100%

In pochissimi casi il modello non riconosce correttamente la domanda ma testando il dataset italiano di *stsb_multi_mt* si è appurato che quando la differenza tra lo score della prima e lo score della seconda frase reputate più simili è maggiore di 0.2, nell'80% dei casi il modello è corretto, quando è minore di 0.2 il modello commette più probabilmente errori. Perciò impostando a 0.2 la soglia di questa differenza, il modello è in grado di comprendere quando la domanda scelta è quella corretta e quando è il caso di proporre all'utente di scegliere tra le prime 3 più simili. Poichè il Rank-3 è del 100% si può affermare che in questo test l'utente fittizio otterrà sempre la risposta alla domanda sottoposta.

Conclusioni e Sviluppi Futuri

Con il progetto di questa tesi si è sviluppata la logica di una chatbot help-desk per un'azienda privata. Durante l'elaborazione sono state affrontate difficoltà legate sia alle dimensioni ridotte del dataset proprietario sia alla quasi totale assenza di dataset pubblici e modelli per la lingua italiana. Come si può evincere dalle alte accuratezze sul Rank-n, attraverso strategie come la data augmentation, la rielaborazione dei dataset multilingua, il fine-tuning e l'apprendimento metrico si è ottenuto un modello abile nel comprendere domande formulate in diversi modi.

Per quanto riguarda gli sviluppi futuri le accuratezze ottenute con i diversi approcci di metric learning suggeriscono che sarebbe opportuno testare diverse rielaborazioni del dataset in quanto la selezione degli esempi informativi è cruciale nei task di classificazione e clustering. La produzione degli esempi è tanto importante quanto la fase di training, avendo il potenziale di migliorare l'accuratezza del modello. Inoltre poichè il dataset multilingua pubblico è stato generato con delle traduzioni, sarebbe anche interessante applicare il metric learning con un dataset in italiano non tradotto.

Ringraziamenti

Ringrazio il mio relatore, Prof. Gianluca Moro, le cui lezioni mi hanno fatto appassionare alla materia al punto da aver illuminato la mia scelta per il corso di Laurea Magistrale. Lo ringrazio inoltre per avermi dato l'opportunità di svolgere questo progetto di tesi e per aver acceso il mio personale interesse nei confronti di questa splendida disciplina.

Ringrazio inoltre il mio co-relatore, dott. Luca Ragazzi, molto disponibile insieme al relatore nel chiarire ogni mio dubbio e ideare insieme soluzioni a problemi sorti durante lo sviluppo del progetto, dandomi preziosi consigli letteralmente a qualsiasi ora del giorno (e della notte).

Bibliografia

[1] Eric A. Weiss. Biographies: Eloge: Arthur lee samuel (1901-90). *IEEE Ann. Hist. Comput.*, 14(3):55–69, 1992.

[2] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, 44(1):206–227, 2000.

[3] Aurelien Geron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Sebastopol, CA, 2017.

[4] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

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

[7] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016.

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

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[10] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[11] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference*

on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 11 2019.

[12] Maxime De Bruyn, Ehsan Lotfi, Jeska Buhmann, and Walter Daelemans. Mfaq: a multilingual faq dataset, 2021.

[13] Edoardo Federici. sentence-bert-base, sentence-transformer for italian, 2022.

[14] Philip May. Machine translated multilingual sts benchmark dataset. 2021.

[15] NLLB Team, Marta R. Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, Anna Sun, Skyler Wang, Guillaume Wenzek, Al Youngblood, Bapi Akula, Loic Barrault, Gabriel Mejia-Gonzalez, Prangthip Hansanti, John Hoffman, Semarley Jarrett, Kaushik Ram Sadagopan, Dirk Rowe, Shannon Spruit, Chau Tran, Pierre Andrews, Necip Fazil Ayan, Shruti Bhosale, Sergey Edunov, Angela Fan, Cynthia Gao, Vedanuj Goswami, Francisco Guzmán, Philipp Koehn, Alexandre Mourachko, Christophe Ropers, Safiyyah Saleem, Holger Schwenk, and Jeff Wang. No language left behind: Scaling human-centered machine translation. 2022.