UNIVERSITÀ DEGLI STUDI DI PERUGIA
Dipartimento di Matematica e Informatica

CORSO DI LAUREA IN INFORMATICA

Tesi di Laurea

I principali modelli di rappresentazione del testo per l'apprendimento automatico

Laureando
Elisa Acciari

Relatore
Prof. Francesco Santini

Anno Accademico 2020/2021

Indice

Introduzione

L'Intelligenza Artificiale (IA) è una scienza fondamentale al giorno d'oggi, che permette alle macchine di apprendere da esempi e dalle proprie esperienze per poi finalizzare il risultato in compiti che emulano l'intelligenza umana. Come dice il gruppo di analytics SAS "è la scienza che mira ad imitare le abilità umane, il machine learning è un sottogruppo specifico dell'intelligenza artificiale che addestra la macchina su come apprendere."[1]

Una branca di questa ampia materia che è l'IA, fa parte il Natural Language Processing (NLP) che supporta i computer nella comprensione, l'interpretazione e l'utilizzo del linguaggio umano.

Per elaborare grandi quantità di dati e creare modelli di apprendimento intelligenti andiamo ad utilizzare degli strumenti che ci facilitano la loro comprensione e il loro sviluppo. Per effettuare previsioni, analizzare e strutturare enormi quantità di dati ricavandone informazioni efficienti e ottimali, ci vengono in aiuto il Natural Language Processing e i suoi strumenti.

La NLP si preoccupa di analizzare e comprendere il linguaggio umano, al fine di elaborarlo per renderlo più semplice e comprensibile per una macchina. Ovvero di tradurlo in un linguaggio che il compilatore riesce a riconoscere per poi utilizzarlo.

L'elaborazione automatica del linguaggio naturale ha lo scopo di implementare strumenti informatici per analizzare, comprendere e generare testi che gli uomini possano comprendere in maniera naturale, come se stessero comunicando con un altro interlocutore umano e non un computer.

Alcuni dei principali settori in cui vengono utilizzate le tecniche di NLP sono ad esempio la traduzione automatica, che oggi è alla portata di tutti online.

Oppure un altro esempio è quello dei cosiddetti predictive text, che sono alla base delle tastiere intelligenti, che permettono di suggerire il completamento della parola che stiamo digitando.

Ancora un altro settore che è una delle ultime innovazioni di questo periodo è l'assistente virtuale e le chatbot con cui possiamo comunicare interagendo in

[1] SAS, Le cinque tecnologie alla base dell'Intelligenza Artificiale (IA), `https://www.sas.com/it_it/insights/articles/analytics/five-ai-technologies.html`

maniera automatica e in tempo reale.

Quindi lo sviluppo di queste tecniche di elaborazione del linguaggio naturale sono state fondamentali per costruire macchine sempre più intelligenti.

Grazie all'Intelligenza Artificiale, che permette alle macchine di imparare dalle proprie esperienze, ad elaborare nuovi dati e svolgere compiti con una logica umana, possiamo ora utilizzare modelli di apprendimento automatico fondamentali per l'elaborazione di tali informazioni.

Infatti, una volta semplificati i dati in linguaggio umano attraverso tecniche di NLP, dobbiamo passarli ad un modello che riesca a elaborarli in un linguaggio comprensibile alla macchina, ad apprendere queste informazioni messe a disposizione e che sia poi in grado di operare su nuovi dati, per eseguire compiti umani.

Nel corso della storia questi modelli si sono evoluti diventando sempre più sofisticati ed accurati.

Tali modelli, come abbiamo largamente spiegato, hanno la capacità di agire su nuovi dati di input fornendo diverse funzionalità in base al tipo di applicazione che si vuole affrontare. Ne andiamo ad elencare alcuni di seguito.

- Classificazione, modellare le relazioni tra i dati e definire la loro classe di appartenenza. Questa è quella che ci interessa e che andremo ad approfondire.
- Regressione, apprende le relazioni funzionali tra le variabili.
- Clustering, raggruppa dati in insiemi omogenei.

Classificazione e Regressione sono esempi di apprendimento supervisionato, ovvero con il quale andiamo a proporre esempi al modello, con i quali dovrà apprendere. Mentre il Clustering è un tipo di esempio di apprendimento non supervisionato.

Inizialmente, nel Capitolo 1, andremo quindi ad introdurre storicamente il Natural Language Processing e lo sviluppo negli anni dei vari modelli di apprendimento automatico, accennando al loro utilizzo pratico.

Per ogni modello trattato introdurremo anche i vantaggi e i lati negativi che hanno portato lo studio e il progresso di modelli migliori e più precisi che si preoccupano di soppiantare i problemi dei modelli precedenti.

Nel Capitolo 2 tratteremo la classificazione, introducendo il concetto di apprendimento supervisionato, ponendo attenzione al lato matematico di un algoritmo di supervised learning. Andremo a spiegare in modo esaustivo i procedimenti e le tecniche dell'elaborazione del linguaggio naturale, per fornire dati comprensibili alla macchina che deve apprendere. Queste tecniche saranno

l'etichettamento dei dati, il preprocessing e l'elaborazione del testo, la trasformazione del testo.

Con la trasformazione del testo andiamo ad introdurre un obbiettivo degli algoritmi di apprendimento automatico, che devono trasformare i dati input in dati numerici in modo da renderli comprensibili per la macchina.

Andremo quindi, nel Capitolo 3, a parlare di uno dei primi modelli utilizzati per questo scopo, Bag of Words, basato sul conteggio delle parole e quindi sulle loro occorrenze nel testo. Proporremo un esempio di elaborazione di frase, proponendo una implementazione del codice.
Andremo poi ad elencare i punti di forza e di debolezza che presenta tale modello.

Nel successivo Capitolo 4 definiremo una implementazione dell'algoritmo precedente, il modello TF-IDF. Al contrario del primo presenterà la differenza che non andrà a contare le occorrenze della parola, ma la frequenza con cui questa appare all'interno del testo, dando maggiore rilievo al significato semantico delle parole.
Anche per questo andremo a trattare un esempio, con implementazione del codice annessa, e a parlare dei vari vantaggi e svantaggi che presenta.

Nel Capitolo 5 andremo a spiegare il modello Word2Vec che risolve i problemi dati dai due precedenti Bag of Words e TF-IDF.
In particolare inizialmente procederemo con un introduzione al concetto di Word Embedding, fondamentale per la costruzione del Word2Vec, e poi andremo a dare una definizione di tale modello.
Spiegheremo la distinzione tra Skip Gram, utilizzato quando in input avremo una sola parola, per ricercare il contesto di cui fa parte, e il Continuos Bag Of Words, il quale attua il procedimento contrario.
Per entrambi andremo a parlare del loro processo di apprendimento, ovvero della Forward Propagation, e della Funzione di perdita Cross-Empty. Quest'ultima sarà utile per andare a calcolare il margine di errore, confrontando il valore atteso con quello realmente ottenuto.
Successivamente andremo a parlare della Back Propagation per ottimizzare l'algoritmo per affinarlo meglio in base obbiettivi che vogliamo ottenere.
Anche per questo metodo andremo a proporre un esempio e i vari vantaggi e svantaggi che presenta.

Infine nell'ultimo capitolo, il Capitolo 7, andremo a spiegare dettagliatamente l'ultima innovazione nel campo dei modelli di apprendimento automatico per l'elaborazione del testo, che sono i Trasformatori, in particolare del modello BERT, che appunto si basa sulla tecnologia dei primi.
Per primo punto andremo ad introdurre i trasformatori e a dare loro una definizione.

Daremo qualche concetto di reti neurali, della loro struttura e del loro scopo, poichè sono una tecnologia precedente ai trasformatori.

Introdurremo anche un particolare tipo di Rete neurale, le Reti neurali Ricorrenti, in particolare della Long Short-Term Memory.

Una volta trattato tutto ciò entreremo nel cuore dei Trasformatori, spiegando la loro struttura, i loro processi, le loro tecniche e tutto quello che ci serve per capirli chiaramente.

Come ultimo argomento, sempre nel Capitolo 7, spiegheremo nel dettaglio il sistema BERT.

Definiremo il suo processo di addestramento, il suo utilizzo pratico, elencando le varie applicazioni in cui viene impiegato.

Infine parleremo dei punti di forza, in particolare dei miglioramenti che porta rispetto a Word2Vec, e dei punti negativi che presenta.

Capitolo 1

Sviluppo nella storia delle tecniche di modellazione per NLP

Secondo gli Osservatori Digital Innovation del Politecnico di Milano[1],

"Per Natural Language Processing o elaborazione del linguaggio naturale si intendono algoritmi di intelligenza artificiale in grado di analizzare, rappresentare e quindi comprendere il linguaggio naturale. Le finalità possono variare dalla comprensione del contenuto, alla traduzione, fino alla produzione di testo in modo autonomo a partire da dati o documenti forniti in input."

Quindi la NLP non è altro che un'area di ricerca dell'Intelligenza Artificiale che si focalizza sull'elaborazione del testo per creare macchine intelligenti e capaci di comprendere il linguaggio umano.

Non solo si deve comprendere il significato di ogni singola parola, ma anche delle varie interpretazioni che può assumere quella parola in quella frase.

Due delle componenti principali del NLP sono la comprensione della lingua naturale (NLU, Natural Language Understanding) e la generazione del linguaggio naturale (NLG, Natural Language Generation).

La prima viene svolta tramite riconoscimento vocale e permette la conversione da lingua naturale ad artificiale.

Come scrive Lorenzo Govoni, nel suo articolo *Elaborazione del Linguaggio Nautrale* nel suo blog, "La maggior parte dei sistemi di riconoscimento vocale oggi sono basati su modelli *Hidden Markov* (HMM), ossia modelli statistici che trasformano il discorso in testo effettuando calcoli matematici per determinare

[1] Osservatori Digital Innovation del Politecnico di Milano, Natural Language Processing (NLP): come funziona l'elaborazione del linguaggio naturale (2021), `https://blog.osservatori.net/it_it/natural-language-processing-nlp-come-funziona-lelaborazione-del-linguaggio-naturale`

ciò che viene pronunciato."[2]

Abbiamo però anche una comprensione reale, che è il passo più difficile della comprensione della lingua naturale, nella quale andiamo ad utilizzare un processo chiamato Part-of-Speech. Di quest'utlimo ne parla sempre L. Govoni nell'articolo citato prima "che consiste nell'etichettare ogni parola secondo una determinata categoria come nome, verbo, avverbio, pronome, e così via .."

La generazione del linguaggio naturale invece è molto più semplice e non è altro che il processo inverso del precedente. Andiamo a codificare il linguaggio artificiale di un computer in testo ed eventualmente si può anche generato un discorso udibile con sintesi vocale.

Quindi una volta datagli in input la richiesta, tramite una domanda, il computer va a determinare quali informazioni tradurre in testo, tramite ad esempio una ricerca online, e ritornerà la risposta.

La NLP comprende alcune fasi che si occupano di superare tali ambiguità e di ridurre il più possibile gli errori. Andiamo ad elencare alcune delle tecniche più comuni che vengono applicate per preparare i dati di testo.

- Tokenizzazione, con il quale andiamo a segmentare il testo in token, in modo da facilitare la conversione dei dati in numeri.
- Rimuovere le stop words e la punteggiatura, quindi eliminare le preposizioni, che possiamo definire come "rumore", ovvero insignificanti al fine che vogliamo.
- Stemming, permette di ridurre un termine dalla forma flessa alla radice, in modo da non considerare parole differenti, che in realtà hanno significato simile come distinte.

Ciò permette di pulire i dati in modo da migliorare la loro leggibilità per il computer.

Il libro *Applied Natural Language Processing in the Enterprise*[10], scritto da Ankur A. Patel e Ajay Uppili Arasanipalai, è una pietra miliare in ambito NLP e tratta tutti i suoi concetti fondamentali come il tokenizer, gli incorporamenti ecc. considerando anche la sua implementazione in Python.

Gli ambiti applicativi dell'elaborazione del linguaggio naturale sono diversi. Alcuni sono riportate qui di seguito.

- Sentiment Analysis, si occupa di comprendere lo stato d'animo con cui una frase viene espressa o scritta.
- Named Entity Recognition, mette in relazione entità specifiche in un testo.

[2]Lorenzo Govoni, *Business e Tecnologia,* , `https://www.lorenzogovoni.com/elaborazione-del-linguaggio-naturale/`

- Traduzione automatica.
- Classificazione dei documenti.
- Chatbot, per comprendere il linguaggio naturale, per gestire dialoghi, ricerca vocale ecc.. Secondo il sito Digital Guide IONS la definizione ufficiale è "[..] un sistema di dialogo testuale. I chatbot dispongono di una maschera di input e output per comunicare in una lingua naturale, simulando teoricamente un interlocutore reale."[3]

L'obbiettivo finale che si vuole raggiungere è quello di creare sistemi intelligenti ed autonomi, in particolare, nel corpo di un robot, in modo da ricreare un intelligenza che meglio si avvicini a quella umana. Fino ad ora abbiamo fatto enormi progressi.

Un esempio lampante è il robot creato dalla IBM di nome Watson, che è persino in grado di individuare alcune emozioni avendo una precisione del'86%[4] .

1.1 Breve Storia

Le origini dell'elaborazione del Linguaggio Naturale partono dagli anni '40, nel secondo dopo guerra, con la Traduzione Automatica.

Nello specifico, queste tecniche ebbero successo grazie allo scienziato statunitense Warren Weaver con un memorandum [1] nel quale veniva proposta per la prima volta l'idea dell'utilizzo di un calcolatore per la traduzione di un testo. Ma ben presto, tra gli anni '60 e '70, la ricerca negli NLP perse considerazione e, a causa dell'inadeguatezza dei sistemi esistenti e della mancanza di dati facilmente fruibili, venne abbandonata.

Grazie alla nascita di Internet e dello sviluppo di risorse in termini di potenza e memoria, negli anni Ottanta ci fu una rapida crescita di questi sistemi NLP.

I primi modelli ad essere utilizzati furono quelli basati sul conteggio delle parole. Il così detto Bag of Words, con il quale otteniamo un vettore contenente le occorrenze di tutte le parole nel testo.

La loro elaborazione del testo però rimaneva superficiale e poco accurata, poichè si ignorava l'ordine in cui apparivano le parole e la grammatica. Nonostante la perdita d'informazione, l'approccio bag-of-words rappresenta una buona strategia per estrarre i concetti chiave da un testo.

[3] Digital Guide IONS, *I chatbot nell'online marketing*, (2020), `https://www.ionos.it/digitalguide/online-marketing/vendere-online/rivoluzionare-lonline-marketing-con-i-chatbot/`

[4] L. Govoni, *Elaborazione del linguaggio naturale*, `https://www.lorenzogovoni.com/elaborazione-del-linguaggio-naturale/`

Nella pratica questo modello viene utilizzato ad esempio come filtro anti-spam. L'idea è quella di considerare che nella lettera di spam saranno presenti più frequentemente parole relative alla vendita come: "buy", "order". Relative alle offerte: "win", "prize", "offer". Ed altre. Possiamo vedere tutte le parole in questione nel sito SqualoMail[5], una piattaforma per l'email marketing.

Mentre lettere normali difficilmente le conterranno. Quindi andrà a contare la frequenza con cui appaiono tali parole nel testo e, in base a queste, classificherà la lettera come spam o meno.

Per ovviare alle problematiche di BOW citate sopra, venne introdotta una sua implementazione chiamata TF-IDF, la quale non utilizza le occorrenze pure delle parole, ma un valore dato dal rapporto tra il numero di volte in cui la parola appare in un documento rispetto al numero totale di parole in quel documento.

Usando questa tecnica, ricompenseremo le parole (aumentando il loro valore di frequenza) che appaiono abbastanza comunemente nel nostro testo, ma raramente in altri testi, mentre puniremo le parole (riducendo il loro valore di frequenza) che appaiono frequentemente sia nel nostro testo che in altri testi come preposizioni, pronomi, ecc..

In termini pratici questo modello viene utilizzato soprattutto nella pratica SEO. Nello specifico, effettua il calcolo delle parole chiave, in modo da poter ottimizzare i contenuti della pagina e posizionarsi ad un livello più alto nella SERP, ovvero nella pagina dei risultati del motore di ricerca.

La svolta si ebbe con i vettori word embedding. In particolare ebbero successo dopo il 2013 con lo studio [2] di un team di Google guidato da Tomas Mikolov, nel quale presentò il modello Word2Vec.

Quest'ultimo è in grado di trasformare le parole in uno spazio vettoriale in modo che parole simili siano rappresentate da vettori simili. Si rilevò molto più efficace nel cogliere le relazioni tra le varie parole e computazionalmente più efficiente rispetto ai precedenti.

I word vectors possono essere utilizzati in vari task supervisionati del NLP, come la classificazione del testo, la named entity recognition e la sentiment analysis.

La named entity recognition "ha lo scopo di identificare, estrarre e classificare automaticamente alcune informazioni chiave presenti nei documenti. Ciò permette di ottenere dei dati strutturati utilizzabili dalle macchine per trovare informazioni, estrarre elementi chiave e anche fatti, domande e risposte."[6]

La Sentiment Analysis "serve a estrarre informazioni a partire da grandi quantità di dati) finalizzate ad analizzare testi "non strutturati", in linguaggio

[5] SqualoMail, *L'elenco delle parole che attivano i filtri della posta indesiderata*, (2019), `https://www.squalomail.com/la-lista-delle-parole-che-attivano-i-filtri-spam/`

[6] Gruppo RES, *Estrarre valore dai dati con il natural Language Processing: la Named Entity Recognition*, (2020), `https://res-group.eu/articoli/estrarre-valore-dai-dati-con-il-natural-language-processing-la-named-entity-recognition`.

naturale, che possiamo ritrovare ad esempio in una email, nella pagina di un sito eCommerce o in un commento su un Social Network. Va a studiare i testi scritti, analizzandone in particolare il livello di positività o negatività, cioè la loro polarità."[7]

Le informazioni semantiche contenute in questi vettori hanno, infatti, dimostrato di incrementare sensibilmente l'accuratezza nella risoluzione di questi task.

Però l'approccio con word2vec non era ottimale. Uno dei principali svantaggi è che non supporta le parole fuori dal vocabolario e quindi non si possono generare rappresentazioni vettoriali fuori dal nostro spazio di parole. Un'altro problema è che generano incorporamenti che sono indipendenti dal contesto e quindi parole uguali, ma in contesti diversi, vengono rappresentate allo stesso modo.

Nel 2017 i ricercatori di Google introdussero i così detti Trasformatori, nel documento [3]*Attention is all you need.*

In particolare, nel 2018 presentarono BERT (Bidirectional Encoder Rappresentations for Transformers) apportando cambiamenti significativi all'elaborazione del linguaggio naturale e risolvendo tutti i problemi di Word2Vec elencati sopra. Questo modello è in grado di riconoscere l'importanza anche delle preposizioni in una frase di ricerca. Ciò migliora i risultati che si ottengono.

Inoltre, anche se sono pochi, si hanno degli impatti positivi sulla SEO.

Successivamente analizzeremo tutti questi modelli in modo specifico e accurato, per avere un idea chiara di come sono fatti e come vengono implementati. Per capire meglio l'elaborazione pratica del linguaggio naturale, possimo fare riferimento al libro di Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta e Harshit Surana, *Practical Natural Language Processing*[12], che tratta proprio di implementare i sistemi NLP in diversi ambienti, come quello aziendale, sanitario e anche dei social media.

[7] D. Avella, *Che cosa è la Sentiment Analysis?*, (2017), `https://www.extrasys.it/it/redblog/che-cosa-%C3%A8-la-sentiment-analysis`.

Capitolo 2

Classificazione

Con la Classificazione testuale andiamo a riferirci ad approcci di classificazione di tipo supervisionato, quindi fornendo esempi al computer completi, da utilizzare come indicazione per eseguire il compito richiesto, con l'obbiettivo di associare contenuti testuali a caratteristiche e classi definite.

2.1 Supervised Learning

L'apprendimento supervisionato è una metodologia di apprendimento automatico nel Machine Learning. Quest'ultimo è una sottobranca dell'Intelligenza Artificiale che si occupa di creare sistemi che apprendono dai dati che gli forniamo. La differenza sostanziale tra Intelligenza Artificiale e Machine Learning è che la prima ha l'obbiettivo di far simulare ad un computer l'intelligenza umana, l'altra consiste di tecniche che consentono ai computer di comprendere e imparare attraverso dati. Ovveo non iniziano in modo intelligente, ma diventano intelligenti.

Nel Machine Learning si suddividono le metodologie di apprendimento in tre categorie:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning
- Semisupervised Learning, in parte supervisionato e in parte non supervisionato.

Quello che ci interessa a noi e che andremo a trattare, come già accennato, sarà l'apprendimento supervisionato.

Il suo obbietivo è quello di generare una funzione, un modello, che sia in grado di apprendere dagli esempi forniti e di produrre poi dei risultati che meglio rispecchiano quelli desiderati, basati su dati nuovi che ancora non conosce. Questo apprendimento differenzia in generale due tipi di algoritmi.

1. Algoritmi di Classificazione, in cui i valori output sono classi discrete, valori qualitativi.

2. Algoritmi di Regressione, in cui i valori output sono valori continui, quantitativi.

Quindi tutti gli algoritmi di apprendimento supervisionato richiedono un numero adeguato di esempi etichettati, detti training set, in modo tale che il sistema accumuli esperienza e impari da questi.

Ovvero, dati degli input x e degli output y, l'algoritmo deve apprendere la funzione f(x) che dall'input genera l'output, tale che

$$y = f(x)$$

L'obbiettivo è quello di creare una funzione h(x), detta funzione ipotesi, in grado di approssimare adeguatamente la funzione f(x), chiamata funzione obbiettivo e che l'algoritmo non conosce, e fornire risposte soddisfacenti.

Il training set sarà utilizzato nella così detta fase di Training, in cui avverrà l'applicazione dell'algoritmo di classificazione, ai fini di derivare regole di classificazione e quindi una funzione di mappatura che assegna una classe a un valore di input. In definitiva va ad elaborare h(x).

Successivamente la macchina deve valutare l'accuratezza della funzione ipotesi, se approssima o meno adeguatamente la funzione obbiettivo. Per capirlo si utilizza un altro insieme dati, detto test set, fornito dal supervisore, il quale rappresenterà esempi diversi dal training set. A questo punto la macchina sviluppa gli output per il test set e poi confronta con la risposta corretta. Il rapporto fra il numero di risposte corrette e il numero di test determina il valore dell'accuratezza.

$$accuratezza = \frac{risposte\_corrette}{numero\ _test}$$

Se la percentuale di quest'ultima è soddisfacente, la funzione ipotesi passa il test. In caso contrario l'apprendimento supervisionato riparte con l'analisi di un ulteriore training set.

Possiamo suddividere la classificazione in tre diversi tipi.

1. Binaria: il numero di classi/etichette a cui vengono associati i contenuti è uguale a due.

2. Multiclasse: abbiamo più di due classi e ogni contenuto può assere associato ad una di queste classi.

3. Multiclasse multilabel: ogni contenuto può essere associato a più di una classe, e quindi avere più di un'etichetta.

Per svolgere la classificazione è necessario inizialmente attuare tre attività fondamentali: etichettare i dati, trasformare ed elaborare il testo.

2.2 Etichettare i dati

Porre un etichetta ad un dato significa andare ad assegnare quella che abbiamo definito come una classe, ad ogni dato input del nostro training set. Questa procedura è fondamentale poichè il nostro modello imparerà a predire i target values, ovvero la classe output, grazie all'associazione tra l'etichetta e l'input del training set.

Si tratta di costruire un training set di contenuti testuali etichettati con le caratteristiche d'interesse.

Possiamo reperire questi dati etichettati direttamente da un file, tipicamente file csv, e porli in una forma leggibile ed utilizzabile per l'addestramento con i modelli di classificazione che introdurremo più avanti.

Il file cvs non è altro che un file di testo che utilizza le virgole per separare i dati contenuti all'interno delle singole celle di una tabella. Ogni riga rappresenterà una diversa istanza della tabella.

Qualora non si abbiano a disposizione dati etichettati, e quindi si sia in possesso solo dei contenuti testuali da utilizzare per l'addestramento, allora si può far ricorso ad annotatori testuali per etichettare i dati; come Doccano, Prodigy o altri annotatori. Una volta annotati i dati possono essere facilmente esportati in formato csv per poi essere letti e manipolati attraverso librerie ad hoc come Pandas. Successivamente si passa a manipolare il testo vero e proprio.

2.3 Trasformazione ed elaborazione del testo

Una volta in possesso dei dati etichettati occorre svolgere delle operazioni sul testo in modo da porre il contenuto testuale in una forma più adatta al task da svolgere (preprocessing) e da renderlo leggibile e analizzabile dagli algoritmi di machine learning che andremo ad utilizzare (trasformazione).

2.3.1 Preprocessing ed elaborazione del Linguaggio Naturale

Si occupa della pulizia del testo, in modo da eliminare il "rumore" ovvero tutte quelle informazioni non rilevani ai fini della classificazione da svolgere. Viene cioè fatta un'elaborazione del Linguaggio Naturale (NLP) che consente di manipolare il testo in una rappresentazione comprensibile per il calcolatore.

Si va quindi ad eseguire una prima operazione chiamata tokenizzazione che si riferisce al dividere le sequenze di caratteri in unità minime di analisi dette "token" e in base a come si effettua la suddivisione avremo un tipo diverso di tokenizer (parole, frasi...).

A seguire possono essere svolte le seguenti azioni:

- Rimuovere le stopwords: eliminare dal testo tutte quelle parti del discorso che nell'ambito di un linguaggio non rappresentano solitamente informazione rilevante, come gli articoli, le proposizioni, le congiunzioni, ecc... Oppure possiamo anche decidere di personalizzare queste parole da eliminare in base alle nostre esigenze.
- Rimuovere la punteggiatura.
- Stemming: consiste nel ridurre un termine dalla forma flessa alla sua radice (running = run), con l'obiettivo di minimizzare l'effetto della presenza di variazioni morfologiche differenti che hanno però lo stesso significato semantico.
- Lowercasing: trasformare tutto il testo in carattere minuscolo in modo da evitare differenziazioni in base al carattere.

Le librarie di NLP come Nltk[1] o Spacy[2] offrono funzionalità già definite per svolgere queste attività.

Una volta terminata la processazione del testo rendendolo "pulito", passiamo alla così detta *Trasformazione del testo*, che consiste nel trasformare i contenuti in modo tale che possano essere letti dagli algoritmi di apprendimento automatico, in quanto non sono in grado di analizzare direttamente le parole testuali, ma operano solo con dati numerici.

Esistono vari algoritmi per codificare i dati di testo, di seguito tratteremo i più importanti e quelli maggiormente utilizzati.

[1] NLTK Documentation, Natural Language Toolkit, `nltk.org`
[2] Spacy, Industrial-Strength Natural Language Processing,`https://spacy.io/`

Capitolo 3

Bag of Words

Il modello Bag of Words, detto anche borsa di parole, è il modello più comunemente usato per la classificazione del testo e non solo. Viene utilizzato anche per il riconoscimento di immagini.

Secondo il data scientist D. Caldarini nell'articolo *Classificazione Testuale attraverso il Machine Learning*

> "Consiste in una rappresentazione vettoriale dove ogni contenuto testuale viene trasformato in un vettore V-dimensionale, con V che rappresenta la cardinalità del vocabolario. In particolare avremo che ogni componente del vettore assumerà valore uguale a zero, qualora il termine non è presente, altrimenti un valore corrispondente al numero di occorrenze di quel termine nel testo."[4]

Questo approccio non presta attenzione all'ordine delle parole e alla grammatica, ma solamente alla frequenza con cui compare tale parola nel testo.

Inoltre si andrà a scegliere un valore limitato di parole da andare ad inserire nel modello, questo proprio per evitare che il set di dati contenga un numero eccessivo di parole inutili e non presenti frequentemente.

3.1 CountVectorized

Per convertire le parole del documento in un set di dati numerici andremo ad utilizzare il modello CountVectorized, che è un implementazione di Bag of Words.

Questo modello ci permette di andare a ripulire il set di dati con il codice *processor=clean* e a generare per ogni parola un vettore di lunghezza pari alla dimensione del vacabolario. Ad ogni parola viene assegnato un numero univoco che sarà rappresentata da un 1 in quella dimensione e con degli 0 nelle restanti.

Ogni volta che incontreremo di nuovo quella parola, aumenteremo il conteggio, lasciando 0 ovunque non abbiamo trovato la parola nemmeno una volta.

Possiamo andare a modificare il valore di max_feature, che utilizzeremo come numero massimo di parole con cui andreamo a creare il vettore, in quanto aggiungerle tutte renderebbe il set dati scarso e con parole inutili, come abbiamo detto in precedenza.

3.2 Esempio

Vediamo un esempio di codice ripreso nell'articolo *CountVectorized in Python*, riportato da Team Edpresso.[1].

```
from sklearn.feature_extraction.text import
    CountVectorizer
text = [''John is a good boy. John watches basketball'']
vectorizer = CountVectorizer(max_features= 50,
    preprocessor=clean)
vectorizer.fit(text)
print(vectorizer.vocabulary_)
vector = vectorizer.transform(text)
print(vector.shape)
print(vector.toarray())
```

Output
'basketball': 0, 'watches': 5, 'good': 2, 'boy': 1, 'is': 3, 'john': 4
(1, 6)
[1 1 1 1 2 1]

Vediamo che il vettore composto dalle occorrenze delle parole nel vocabolario corrispondente al documento "John is a good boy. John watches basketball" è:

[1 1 1 2 1]

in cui la parola "jhon" compare due volte, mentre le restanti tutte una sola volta. In questo caso il numero massimo di parole da riportare sarà 50 e il vettore corrispondente ad esempio alla parola "jhon sarà:

[0 0 0 1 0]

Quindi tutte le parole univoche che compaiono nel documento di input vengono racchiuse in un "sacco" ovvero un vettore, con dimensione pari a quella del vocabolario, che viene riempito con il numero di volte in cui le diverse parole

[1] Teaam Edpresso, *CountVectorized in Python*, `https://www.educative.io/edpresso/countvectorizer-in-python`

appaiono nel testo.

3.3 One Hot Encoding

Il problema di Count vector è che token che appaiono frequentemente sono rappresentati più significativi rispetto agli altri.

Una soluzione è utilizzare la codifica one-hot che utilizza un vettore booleano per indicare la presenza o l'assenza di un token all'interno di un documento, rispettivamente con un 1 o con uno 0. Questo riduce il problema dello sbilanciamento della distribuzione dei token.

Per l'esempio precedente, i vettori one hot sono, per ogni, parola i seguenti.

basketball: [1 0 0 0 0 0]

watches: [0 1 0 0 0 0]

good: [0 0 1 0 0 0]

boy: [0 0 0 1 0 0]

is: [0 0 0 0 1 0]

john: [0 0 0 0 0 1]

Vediamo che ognuna ha una rappresentazione vettoriale di dimensione uguale alla dimensione del vocabolario, in cui ogni vettore differisce dalla posizione del numero 1 che presenta. Il risultato precedente mostra il vettore contenente la somma delle parole presenti nella frase di input. Infatti avremo due occorrenze della parola "good" e l'output presenterà il valore "2" nella posizione dell'indice "1" di quella parola.

Una codifica a caldo è un metodo di conversione dei dati per prepararlo per un algoritmo e ottenere una previsione migliore.

3.4 Vantaggi e Svantaggi

Nonostante sia un modello relativamente semplice, i punti di forza di Bag of Words risiedono nella sua semplicità: è poco costoso da calcolare e si presta meglio nei casi in cui il posizionamento o le informazioni contestuali non sono rilevanti.

Tuttavia presenta comunque vari svantaggi.

– L'inserimento di nuove frasi causerebbe un aumento della grandezza del vocabolario e quindi dei vettori.

– Più aumentiamo il vocabolario, più saranno presenti valori pari a 0 nel vettore e ciò darebbe vita ad una matrice sparsa, la cui rappresentazione comporterebbe una grande quantità di memoria.

– Inoltre, il BoW non fornisce informazioni riguardo la grammatica, la posizione e soprattutto il significato di una parola. Questa mancanza rappresenta un problema non indifferente nella progettazione di modelli e applicazioni sempre più sensibili alle specificità del linguaggio umano.

Capitolo 4

TF-IDF

Come riportato da D. Caldarini nell'articolo citato prima [4]

> "Tf-Idf sta per Term Frequency/Inverted Document Frequency e si tratta di una rappresentazione vettoriale simile a quella del metodo Bag of Words."

Infatti è appunto un'estensione di quest'ultimo. La novità che apporta questa tecnica è quella di utilizzare, anzichè le occorrenze della parola, un valore Tf-Idf definito come il rapporto tra la Term Frequency, ovvero la frequenza del termine all'interno del documento, e la Inverse Document Frequency, una misura di quanto il termine occorre all'interno dell'intero corpus (in quanti documenti essenzialmente).

La Frequenza del Termine, ovvero la Term Frequency (tf), ci da appunto la frequenza della parola all'interno del documento. È definita come il rapporto tra il numero di volte in cui la parola appare in un documento ($n_{i,j}$) rispetto al numero totale di parole in quel documento, che indicheremo con D.

Aumenta all'aumentare del numero di occorrenze di quella parola nel testo.

$$tf_{i,j} = \frac{n_{i,j}}{D}$$

La Inverse Document Frequency (idf), invece, è una misura che da maggiore peso alle parole rare e specifiche nel testo e quindi avranno un valore idf più alto, mentre e basso nelle parole comuni. è definito come il logaritmo del numero di documenti (N) diviso per il numero di documenti che contengono la parola w (d_i), in questo caso.

$$idf(w) = log(\frac{N}{d_i})$$

Il valore finale del TF-IDF è il prodotto fra queste due misure.

$$tf.idf_{i,j} = tf_{i,j} * log(\frac{N}{d_i})$$

La formula che TF-IDF utilizza per calcolare l'importanza di una parola ha lo scopo di preservare le parole che sono le più frequenti e le più ricche semanticamente. Più è alto questo valore, più tale parola sarà importante.

4.1 Esempio

Prendiamo l'esempio svolto su ichi.pro[1]. Abbiamo tre documenti:

1. It's raining today
2. I'm not going out today
3. I'm going to watch the premiere of the season.

Una volta puliti e tokenizzati i dati andiamo a calcolarci i vari valori relativi a tf, idf. Nella Tabella 4.1 possiamo vedere i relativi valori calcolati per ciascuna parola in ogni documento.

Parole	tf1	tf2	tf3	idf
going	0.16	0.16	0.12	log(3/3)=0
to	0.16	0	0.12	log(3/2)=0.41
today	0.16	0.16	0	log(3/2)=0.41
i	0	0.16	0.12	log(3/2)=0.41
am	0	0.16	0.12	log(3/2)=0.41
it	0.16	0	0	log(3/1)=1.09
is	0.16	0	0	log(3/1)=1.09
rain	0.16	0	0	log(3/1)=1.09

Tabella 4.1: Valori tf e idf per ogni parola nei vari documenti.

Costruiamo poi la Tabella 4.2 con tutte le parole con i relativi valori tf-idf e andiamo a confrontare i risultati.

[1] ichi.pro, *TF-IDF / Term Frequency Technique: la spiegazione più semplice per la classificazione del testo in NLP con Python.*, `https://ichi.pro/it/tf-idf-term-frequency-technique-la-spiegazione-piu-semplice-per-la-classificazione-del-testo-in-nlp-con-python-154634143291587`

	going	to	today	i	am	it	is	rain
Documento1	0	0.07	0	0	0	0.17	0.17	0.17
Documento2	0	0	0.07	0.07	0.7	0	0	0
Documento3	0	0.05	0	0.05	0.05	0	0	0

Tabella 4.2: Valori tf-idf per ogni parola in ogni documento

Vediamo che per il Documento1 le parole più rilevanti sono "t" ,"is" e "rain", mentre quelle meno anti sono quelle relative al valore *tf-idf=0*.

Un modo molto semplice di implementare questo modello in python è utilizzando la libreria di scikit-learn: *sklearn.feature_extraction.text*.

Il codice *.fit_transform* impara il vocabolario e il tf-idf, restituisce la matrice documento-termine. Pandas per poi creare un DataFrame dal vettore tf-idf risultante, impostando il valore TF-IDF come colonna e le parole come righe.

```
 pandas as pd
from sklearn.featur_extraction.text  TfidfVectorizer
vectorizer = TfidfVectorizer(input='documento')
X = vectorizer.fit_transform(corpus)
df= pd.DataFrame(X.toarray(), index=X.get_feature_names()
    , columns=["TF-IDF"])
df = df.sort_values('TF-IDF', ascending=False)
print (df.head(25))
```

4.2 Vantaggi e Svantaggi

Elenchiamo i vantaggi del modello TF-IDF di seguito.

- In conclusione il modello TF-IDF contiene informazioni sulle parole più e meno importanti, quindi le parole chiave di un documento, rispetto a Bag of Words che crea semplicemente un insieme di vettori contenenti le occorrenze delle parole nel documento.
- Anche se BGW è molto semplice da implementare, l'altro di solito offre prestazioni migliori nei modelli di apprendimento automatico.

Tuttavia uno svantaggio di entrambi è il problema della comprensione del contesto delle parole e la somiglianza tra di esse.

Andiamo ora ad introdurre un nuovo modello che si propone come soluzione ai problemi sopra elencati.

Capitolo 5

Word2Vec

5.1 Word Embedding

I word embedding, tradotti in italiano come Incorporamenti di parole, sono dei metodi di rappresentazione del testo in cui i dati testuali vengono convertiti in vettori numerici, garantendo che parole con significati simili abbiano rappresentazioni numeriche simili. È un processo che avviene in due fasi: la Tokenizzazione e la Vettorializzazione.

La prima, come abbiamo visto, si occupa della pulizia del testo per la costruzione del dataset. La seconda si occupa di mappare ogni parola in un vettore in base alle caratteristiche di quella parola.

Nel nostro spazio vettoriale quindi, i vettori rappresentati le parole si troveranno in posizione vicina a quelle affini e il punto in cui si trovano ci dirà le loro caratteristiche.

Quindi, gli incorporamenti di parole sono più efficaci nel catturare il contesto delle parole all'interno di una frase, al contrario degli algoritmi visti in precedenza in cui le rappresentazioni delle parole non catturavano in alcun modo la somiglianza tra di esse.

Nella Figura 5.1, riportata nell'articolo del Data Scientist A. Sen [5], possiamo vedere un esempio di incorporamento di parole. Ad esempio la parola domenica, "sunday", avrà valori più simili ad altri giorni della settimana rispetto ai membri di una famiglia.

Notiamo infatti che la parola in questione appare nello spazio vettoriale vicino alle altre parole della settimana.

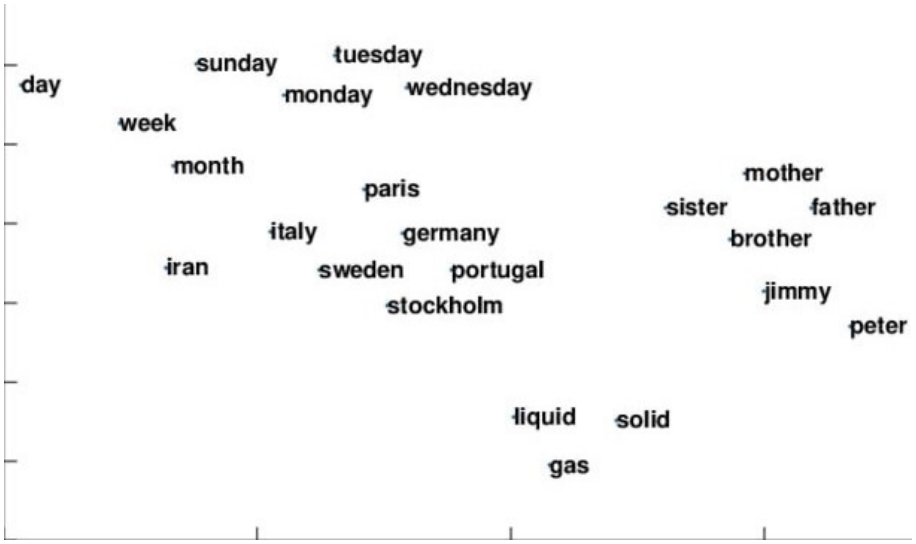

Figura 5.1: Word Embedding

Esistono diversi algoritmi e approcci utilizzati per creare word embedding. Alcuni dei metodi di incorporamento di parole più comuni e affidabili includono: word2vec, livelli di incorporamento e GloVe. Quello che ci interessa e che andremo a trattare di seguito sarà il Word2Vec.

5.2 Word2Vec

Come viene spiegato in uno dei post in ihal.it, un sito di articolo sull'intelligenza artificiale "Word2Vec è stato sviluppato dai ricercatori di Google ed è uno dei metodi di incorporamento più comunemente usati, poiché produce in modo affidabile incorporamenti utili e ricchi. Le rappresentazioni Word2Vec sono utili per identificare i punti in comune semantici e sintattici nel linguaggio. Ciò significa che le rappresentazioni di Word2Vec catturano le relazioni tra concetti simili."[1].

Questi word embeddings, come abbiamo detto, sono delle rappresentazioni vettoriali delle parole. Queste ultime saranno appunto in grado di identificare i punti in comune semantici e sintattici del linguaggio.

Il modello legge le parole codificate come vettori della stessa dimensione attraverso la rappresentazione one-hot-encoding spiegata nel Capitolo 3, Bag of Words.

[1] ihal.it, *Come funziona la classificazione del testo?*, `https://ihal.it/come-funziona-la-classificazione-del-testo/`

L'obbiettivo è quello di addestrare i Word Embedding tramite l'utilizzo di una rete neurale completamente connessa con un singolo livello nascosto.

Il livello di input è impostato per avere tanti neuroni quante sono le parole nel vocabolario. La dimensione del livello nascosto sarà la dimensione dei vettori di parole risultanti. La dimensione del livello di output è la stessa del livello nascosto.

Avremo difatti un primo livello di input, da questo poi il vettore di input viene elaborato nell'hidden layer secondo delle caratteristiche. Infine l'output viene sviluppato attraverso la funzione Softmax, che serve per ottenere la probabilità di distribuzione delle parole. Ne parleremo più approfonditamente nella sezione 5.5, ma in definitiva viene utilizzata in modo che ogni elemento del vettore di output descriva la probabilità che una parola specifica appaia nel contesto.

Quindi il livello di output altro non è che un classificatore Softmax. In particolare, ogni neurone di output ha un vettore di peso che viene moltiplicato con il vettore del livello nascosto, successivamente viene applicata la funzione exp(x) e infine dividiamo il risultato per la somma dei risultati di tutti i nodi output, come possiamo osservare nella Figura 5.2

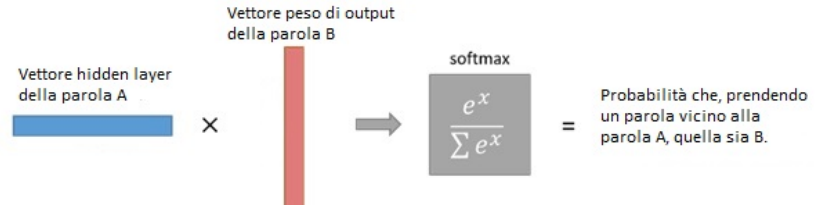

Figura 5.2: Applicazione funzione Softmax

Questo addestramento viene effettuato cercando di ottimizzare una funzione obbiettivo che ha il compito di effettuare la predizione dell'output. Come vedremo poi possiamo avere due tipi di output: una certa parola centrale o un insieme di parole che rappresentano il contesto della parola input all'interno di una finestra.

Dato un documento di lunghezza L, con w_t la parola centrale e D la dimensione delle finestre, osserviamo nella Figura 5.3 la loro rappresentazione.

Figura 5.3: Parola centrale e finestra di contesto

Esistono due metodi in Word2Vec per apprendere le rappresentazioni delle parole:

1. Skip-gram, dove una singola parola è usata per predire il contesto, ovvero prevede le parole all'interno di un determinato intervallo prima e dopo la parola corrente nella stessa frase.
2. Continuous Bag of Words (CBOW), il cui obbiettivo è predire una parola target in base al contesto circostante.

5.3 Skip Gram

Il modello Skip Gram si occupa dell'addestramento di word Embeddings attraverso l'ottimizzazione di una funzione obbiettivo che, partendo da una parola correte w_t, calcola la probabilità delle parole di contesto in una certa finestra.

5.3.1 Forward propagation

Con forward propagation si intende la propagazione delle informazioni in avanti: dal livello di input a quello di output.

Se il vocabolario è costituito da V parole e D è la dimensione dei vettori, ovvero la dimensione della finestra, l'ingresso della connessione tra l'input con lo strato nascosto è rappresentato da una matrice W_1 di dimensione VxD con ciascuna riga che rappresenta una parola del vocabolario.

Poichè i vettori sono in codifica one-hot, moltiplicare un vettore input per la matrice W_1 equivale semplicemente a selezionare una riga di questa. Quindi uno solo dei nodi di input avrà un valore diverso da zero. Ciò significa che per una parola verranno attivati solo i pesi associati al nodo di ingresso con valore 1. A questo punto possiamo dire che anche nei nodi nascosti verranno attivati solo i pesi collegati a quel nodo di input.

Allo stesso modo, le connessioni dallo strato nascosto a quello di uscita sono descritte dalla matrice W_2 di dimensione DxV. In questo caso, ogni colonna della matrice W_2 rappresenta una parola del vocabolario.

L'output che otteniamo viene passato attraverso la funzione Softmax e successivamente i vettori ottenuti li andremo a confrontare con le parole di contesto per ottenere la funzione di perdita.

Per avere risultati ottimali dovremo però modificare queste matrici dei pesi in modo di avvicinarci quanto più possibile all'output atteso. Questo procedimento viene effettuato con la Back Propagation, che descriveremo nella Sezione 5.6,

che va a minimizzare la funzione di perdita. Le matrici W_1 e W_2 vengono chiamate matrici dei pesi.

Dal livello nascosto al livello di output la matrice dei pesi W_2 viene utilizzata per calcolare il punteggio per ogni parola nel vocabolario e la funzione Softmax serve per ottenere la probabilità di distribuzione delle parole di contesto risultanti.

L'output restituisce un singolo vettore contenente, per ogni parola del vocabolario, la probabilità che una parola appaia nel contesto. Più questi risultati sono vicini agli obbiettivi previsti, migliori sono le prestazioni della rete neurale e minore è la funzione di perdita.

Nella Figura 5.4, ripresa dall'articolo di A. Sen in Medium [5], riportiamo il procedimento visivamente. Partiamo dal vettore di input di dimensione V fino ad arrivare al risultato output, ovvero la probabilità che ciascuna parola appaia nel contesto della parola input.

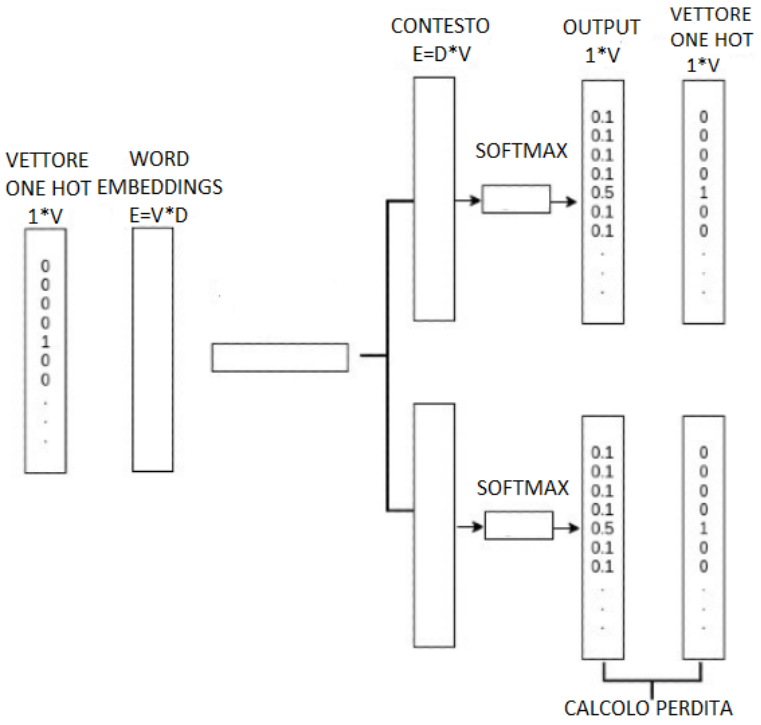

Figura 5.4: Modello Skip Gram

5.4 Modello CBOW

Data la Bag of Words delle parole di contesto w_{t-j}, l'obbiettivo di questo modello è quello di predire una parola centrale, basandosi su una sequenza contigua di n-grammi. w_t, attraverso l'ottimizzazione di una funzione obbiettivo.

Ricordiamo che con l'utilizzo di Bag of Word andiamo a riportare l'informazione sulla molteplicità delle parole che appaiono nel contesto, ignorando l'ordine e la grammatica.

5.4.1 Forward Propagation

Per il modello CBOW, questa fase è identica a quella per Skip Gram, ma con la procedura inversa.

A livello input avremo i nostri vettori di dimensione V, ovvero la dimensione del vocabolario, che rappresentano le parole di contesto. Il valore del livello nascosto viene calcolato moltiplicando poi per D, ottenendo le matrici dei pesi, e facendo la media dei vettori di ingresso pesati.

Nello stato output viene effettuata la funzione Softmax, come per Skip Gram, per ottenere la probabilità che una parola combaci il risultato che vogliamo. L'output restituisce un singolo vettore contenente le probabilità che ogni parola si verifichi.

Viene poi confrontato con il vettore one-hot del linguaggio e viene calcolata la funzione di perdita. Nella Figura 5.5, ripresa sempre a dall'articolo di A. Sen in Medium [5], riportiamo visivamente il processo.

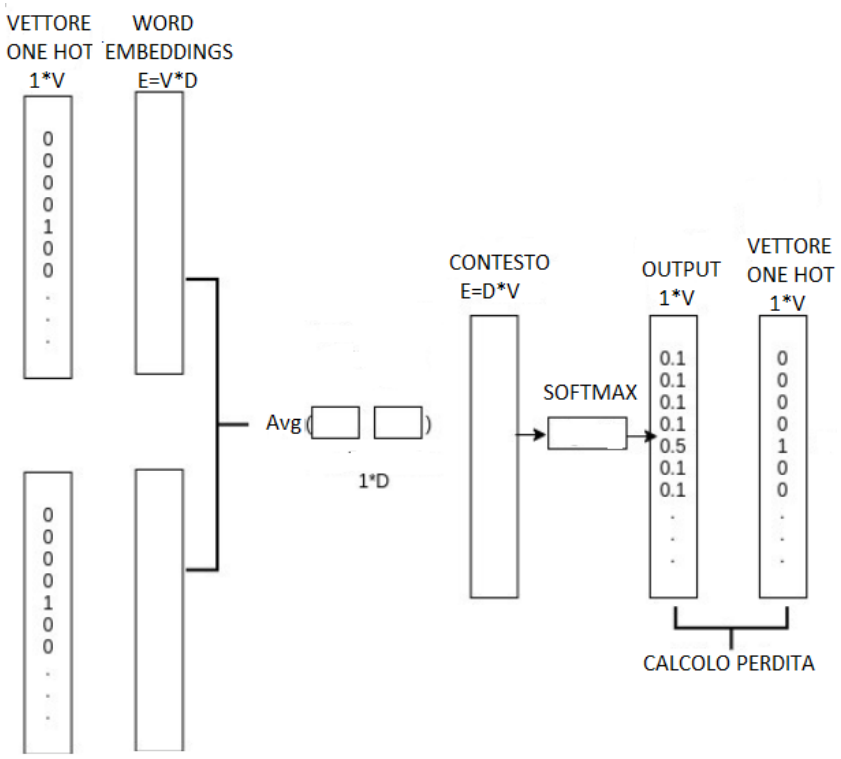

Figura 5.5: Modello Continuous Bag of Words

Anche in questo caso dobbiamo effettuare la Back Propagation per ottimizzare la nostra rete e minimizzare la funzione di perdita.

5.5 Softmax e Funzione di perdita Cross-Empty

Ciò che fa una funzione di attivazione Softmax è prendere un vettore di input di dimensione N e di regolare i valori in modo tale che ogni valore sia compreso tra zero e uno.

$$S(x) = \frac{e^x}{\sum e^x}$$

La proprietà interessante di questa funzione è che la somma di tutte le uscite di softmax, è sempre uguale a 1.

$$\sum S(x) = 1$$

Infine, per quanto riguarda la funzione Cross-Empty, tradotta in italiano come errore di entropia incrociata, viene descritta in un articolo su ichi.pro nel seguente modo.

> "Ogni probabilità di classe prevista viene confrontata con l'output 0 o 1 desiderato della classe effettiva e viene calcolato un punteggio / perdita che penalizza la probabilità in base alla distanza dal valore atteso effettivo. La penalità è di natura logaritmica e produce un punteggio elevato per grandi differenze vicine a 1 e un punteggio basso per piccole differenze tendenti a 0."[2]

La rappresentazione matematica di tale funzione tra due distribuzioni discrete p e q, per n classi, è la seguente e viene utilizzata quando si regolano i pesi del modello durante l'addestramento:

$$L = -\sum_{i=1}^{n} p_i * log(q_i)$$

In questo caso p (fissato) è il vettore target, ovvero il valore atteso, mentre q il vettore di output della rete, ovvero l'output della nostra funzione softmax *S(x)*.

Il valore minimo di L (sempre maggiore di 0) si ha quando il vettore target coincide con l'output.

Ora che abbiamo un'espressione per la funzione di perdita vogliamo trovare i valori di W_1 e W_2 che la minimizzano.

5.6 Backpropagation

La Backpropagation è un algoritmo efficiente che viene utilizzato per trovare i pesi ottimali di una rete neurale: quelli che minimizzano la funzione di perdita.

Per farlo dobbiamo applicare l'algoritmo di discesa del gradiente per modificare i valori dei pesi al fine di ottenere una misura più desiderabile per la probabilità calcolata.

Questo algoritmo implica la ricerca delle derivate della funzione di perdita rispetto ai pesi.

5.6.1 Ottimizzazione basata su gradiente

È l'approccio più utilizzato per implementare Backpropagation. La discesa del gradiente è un metodo di ottimizzazione per trovare il minimo di una funzione.

Considerando il grafico nella Figura 5.6, ripresa dall'articolo di Zafar Ali[3], che raffigura la parabola il cui punto di minimo è l'obbiettivo della discesa del gradiente.

[2] ichi.pro, *Funzione di perdita di entropia incrociata*, `https://ichi.pro/it/funzione-di-perdita-di-entropia-incrociata-267783942726718`

[3] Z. Ali, *A simple Word2vec tutorial*, Medium, (2019), https://medium.com/@zafaralibagh6/a-simple-word2vec-tutorial-61e64e38a6a1

L'idea è di svolgere passi ripetuti verso il minimo nella direzione opposta. Dobbiamo quindi calcolare il gradiente che verrà utilizzato per scegliere la direzione in cui compiere un passo per spostarsi verso l'ottimo locale, attraverso il calcolo di derivate.

In matematica, la derivata è la pendenza di una funzione in un determinato punto; il gradiente è la stessa cosa, ad eccezione del fatto che è un vettore e ognuno dei suoi componenti è una derivata parziale rispetto a una specifica variabile.

Se per esempio quindi avremo più di una variabile, dovremo calcolarci le varie derivate in funzione di ognuna di queste.

L'obbiettivo quando andiamo ad utilizzare il gradiente è minimizzare la funzione. Per farlo avremo bisogno di conoscere il tasso di apprendimento, che rappresenta quanto ci si muove nel nostro grafico in Figura 5.6, ovvero rappresenta la velocità di apprendimento.

Con un alto tasso possiamo coprire più terreno ad ogni passo, quindi questo meccanismo è ottimo in termini di tempo, ma rischiamo di saltare il punto più basso.

Con un basso tasso invece accade il contrario; è molto più sicuro rispetto al punto di minimo, ma il calcolo del gradiente richiederà più tempo e quindi anche per arrivare infondo alla nostra curva.

L'algoritmo va a moltiplicare il gradiente per questo numero, che rappresenta la velocità di apprendimento, o meglio la larghezza del passo da compiere, per determinare il punto successivo. La direzione in cui sarà il passo successivo sarà quella opposta al gradiente. Così facendo andiamo a trovare il punto di minimo, dove il costo della funzione ha raggiunto il suo minimo.

Il peso verrà modificato facendo un passo nella direzione del punto ottimale, che rappresenta il punto più basso della parabola.

Il nuovo valore viene calcolato sottraendo dal valore del peso corrente la funzione derivata nel punto del peso.

Dopo vari passaggi, quando si arriva ad un situazione in cui il costo non migliora e si arriva sempre vicino ad un determinato punto, abbiamo quella che è definita come convergenza. I valori dei parametri in questo ultimo step sono noti come i "migliori".

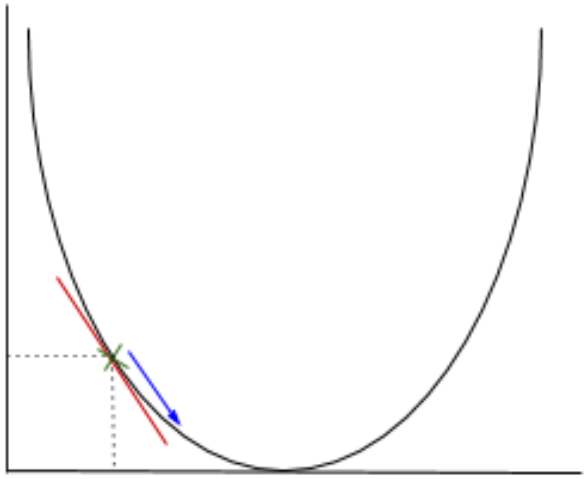

Figura 5.6: Funzione parabolica e inclinazione del gradiente

La discesa del gradiente stocastico aggiorna i pesi sottraendo il peso corrente di un fattore $\propto$, cioè il tasso di apprendimento del suo gradiente.

$$w_{new} = w - \propto \frac{\partial L}{\partial w}$$

Dove: w_{new} sono i pesi da aggiornare.
w sono i pesi che dobbiamo modificare.
L è la funzione di perdita.
∂w decide come modificare questo peso per diminuire la perdita.
$\propto$ è la velocità di apprendimento, ovvero la larghezza del passo da compiere.

Il problema della discesa del gradiente è che in ogni fase viene utilizzato l'intero set di allenamento. Tale calcolo è costoso su un set di dati molto grande.

In questi casi si preferisce utilizzare la *discesa del gradiente stocastico* che indica un sistema collegato a una probabilità casuale. Quindi utilizza un solo campione che viene selezionato in modo casuale per ogni iterazione, invece dell'intero set di dati.

In particolare, l'indice del dato è scelto tramite un campionamento con reinserimento, ossia, dopo aver scelto e utilizzato un campione, esso viene reintrodotto nel training set e quindi nel passaggio successivo può essere scelto con la stessa probabilità degli altri punti.

Questo significa che richiederà un numero maggiore di iterazioni per raggiungere i minimi rispetto al tipico Gradient Descent, detto anche Batch. Rimane comunque ancora computazionalmente meno costoso di Gradient Descent.

Quindi la differenza sostanziale tra i due modelli è che il Batch aggiorna i pesi dopo aver valutato tutti i campioni di allenamento, nello Stocastico, i pesi vengono aggiornati dopo ogni campione di allenamento.

In questo modo abbiamo modificato i pesi dei vettori al fine di avvicinarci quanto più possibile al risultato atteso.

5.7 Esempio

Riportiamo l'esempio fatto da Zhi Li, studentessa in Data Science, in un articolo su Meduim[4]. Andremo ad analizzare la tabella dati relativa a dei modelli di macchina riportata nella Figura 5.7, sempre ripresa dallo stesso articolo dell'esempio.

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size
				and output; double click to hide output							
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	Compact
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact

Figura 5.7: Tabella contenente dati di macchine

Una volta estratti e processati i dati da questa tabella, andiamo ad analizzarne i vari valori con il modello Word2Vec.

```
 gensim
from gensim.models  Word2Vec
#addestriamo il nostro modello
model = Word2Vec(data, min_count=1,size= 50, window =3,
    sg = 1)
```

- *size,* andiamo ad indicere la dimensione degli embeddings.
- *window*, le massime parole di una finestra di contesto, o meglio la distanz dalla parola target.
- *min_count*, le parole con occorrenza inferiore a questo conteggio verranno ignorate.

[4]Zhi Li, *A Beginner's Guide to Word Embedding with Gensim Word2Vec Model*, in Medium,(2019), `https://towardsdatascience.com/a-beginners-guide-to-word-embedding-with-gensim-word2vec-model-5970fa56cc92`

- *sg*, rappresenta l'algoritmo di addestramento. *CBOW=0* e *Skip gram=1*.

Per ottenere la somiglianza fra due modelli di marca ci basterà semplicemente richiamare *model.similarity()* come segue.

```
model.similarity('Porsche 718 Cayman', 'Nissan Van')
model.similarity('Porsche 718 Cayman', 'Mercedes-Benz SLK
    -Class')
```

5.8 Differenza tra Skip Gram e CBOW

Secondo Mikolov, citato nell'articolo di Yanchuan Sim pubblicato sul sito Quora[5], la seguente è la principale differenza tra i due modelli.

- Skip-gram: funziona bene con una piccola quantità di dati di allenamento, rappresenta anche parole o frasi rare.
- CBOW: molte volte più veloce da addestrare rispetto al primo, una precisione leggermente migliore per le parole frequenti.

5.9 Vantaggi e Svantaggi

Elenchiamo di seguito i vantaggi che mostra Word2Vec.

- Word2Vec riesce a rappresentare parole simili attraverso il loro contesto, a differenza dei modelli Bag-of-Words e TF-IDF, che si interessano solo al numero con cui le parole si presentano nel testo.
- Ma il vantaggio principale dell'approccio word2vec è la capacità di apprendere word embedding di alta qualità con poca complessità di spazio e tempo.

Word2Vec presenta però molti svantaggi.

- Gli incorporamenti generati sono indipendenti dal contesto; ciò significa che diversi significati della stessa parola sono combinati in un unico vettore. Ovvero non riconosce le diversità semantiche della parola.
- Non tengono conto della posizione della parola.
- Non supporta le parole fuori dal vocabolario, ciò viene chiamato *Out Of Vocabulary (OOV)*. Ovvero il modello non riesce a generare vettori di incorporamento per parole che non risiedono già nel vocabolario.

[5] Y. Sim, *What are the continuous bag of words and skip-gram architectures?*, Quora, (2017), `https://www.quora.com/What-are-the-continuous-bag-of-words-and-skip-gram-architectures`

Capitolo 6

Trasformatori e modello Bert

I Trasformatori, detti Transformer in inglese, "sono potenti modelli di apprendimento profondo che possono essere utilizzati per un'ampia varietà di attività di elaborazione del linguaggio naturale."[1]

Vengono introdotti nel documento "Attention is all you need" pubblicato dai ricercatori della Google. Per parlarne però prima si deve introdurre il concetto di Attenzione, spiegato sempre nel testo sopra citato.

"Il meccanismo dell'attenzione esamina una sequenza di input e decide ad ogni passaggio quali altre parti della sequenza sono importanti."[2] Il funzionamento è simile al processo di lettura: mentre leggiamo ci concentriamo sulla parola che stiamo leggendo in quell'istante, ma la mente conserva anche le parole passate che sono chiave per capire il contesto della frase.

I trasformatori vengono definiti come "modelli di apprendimento automatico semi-supervisionati che vengono utilizzati principalmente con dati di testo e hanno sostituito le reti neurali ricorrenti nelle attività di elaborazione del linguaggio naturale."[3]

6.1 Reti Neurali

Un altro algoritmo molto adatto a svolgere attività di elaborazione del linguaggio naturale, come task di classificazione testuale, è quello delle reti neurali.

[1] ichi.pro, *Cosa sono i trasformatori e come puoi usarli.*, `https://ichi.pro/it/cosa-sono-i-trasformatori-e-come-puoi-usarli-272459123525402`

[2] ichi.pro, *Cos'è un trasformatore?*, `https://ichi.pro/it/cos-e-un-trasformatore-229238812437508`

[3] ichi.pro, *Cosa sono i trasformatori e come puoi usarli?*, `https://ichi.pro/it/cosa-sono-i-trasformatori-e-come-puoi-usarli-272459123525402`

Le reti neurali sono basate sull'idea di riprodurre l'intelligenza ed in particolare l'apprendimento, simulando la struttura neurale del cervello.

Sono divise in Layer (livelli), ognuno dei quali è composto da una serie di Neuroni. I Neuroni non sono collegati tra loro, ma ogni Neurone è connesso a tutti i Neuroni del livello successivo e di quello precedente.

Un libro interessante da prendere in considerazione per capire meglio questi sistemi è *Deep Learning An MIT Press book* [13], di Ian Goodfellow, Yoshua Bengio e Aaron Courville, il quale parla in particolare del Deep Lerning, delle nozioni base sull'apprendimento automatico e delle reti neurali.

Una rete neurale è una rete divisa in tre livelli.

- Input Layer: in questo livello vengono trattati i dati attraverso una prima funzione (matematica) di attivazione.

  Tale funzione non fa altro che combinare i valori che vengono dati in input ad un neurone ed elaborarli in output. Con la funzione di attivazione decidiamo se far passare l'informazione tramite l'assone, agli altri neuroni oppure no.

- Hidden: eseguono una combinazione lineare (somma pesata) dell'output ottenuto dall'Input Layer; l'output di questi neuroni è una funzione di attivazione della combinazione; in una rete neurale possono esser presenti più hidden layer. Quelle "tradizionali" contengono 2-3 layer, mentre le reti neurali profonde possono contenerne oltre 150.

- Output: stesso funzionamento dell'hidden layer. Sono il livello finale della rete e l'output di questi neuroni è il risultato dell'intera rete.

Figura 6.1 ripresa da un documento del professore Polo Medici, dell'Università degli Studi di Parma [4]

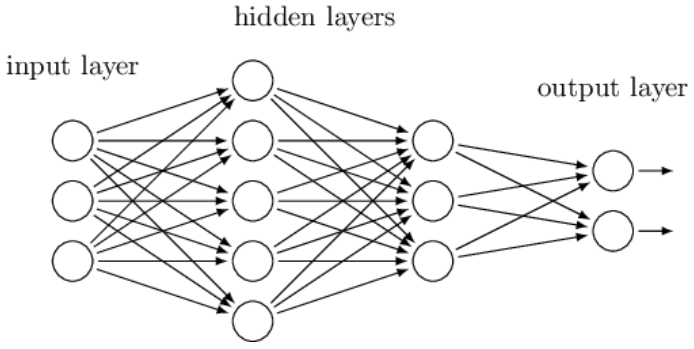

Figura 6.1: Strati di una rete neurale

[4] P. Medici, Università degli Studi di Parma, *Reti neurali*, `http://www.ce.unipr.it/~medici/geometry/node107.html`

Vediamo ora come sono fatti i neuroni.

- $x_1..x_n$ sono i segnali in ingresso al neurone, che possono essere gli input del problema o le uscite di altri neuroni.
- Pesi o Sinapsi w_i: ogni input viene pesato tramite il peso della connessione; fornisce una misura di quanto "conta" tale input nel neurone.
- Sommatore : modulo che effettua la somma pesata degli ingressi.
- Funzione di attivazione g: funzione che determina l'output del neurone sulla base dell'uscita del sommatore.
- Riassumendo, l'output O del neurone si ottiene con

$$O = g(\sum_{i=1}^{n} w_i * x_i)$$

Possiamo dire che: se la somma pesata degli input è maggiore di una certa soglia t, allora il neurone "si accende" (output 1), altrimenti no.

La Figura 6.2 illustra l'architettura di un neurone partendo dai valori di input fino all'output dato dalla funzione di attivazione.

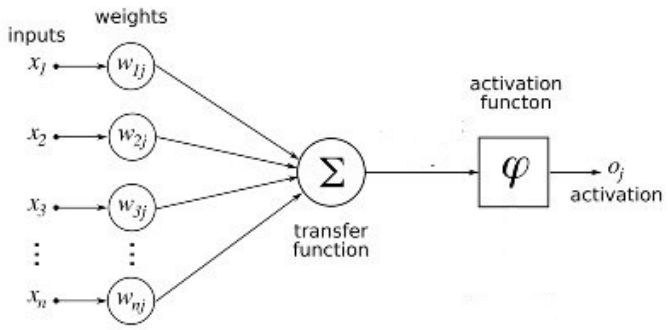

Figura 6.2: Neurone di una rete neurale

Quindi cosa succede?

1. L'ingresso x_i viene moltiplicato per la matrice dei pesi w_i associata ad un certo neurone.

2. Successivamente questa informazione viene fatta passare per una funzione di attivazione e si continua così per ogni Layer. La funzione di attivazione in uscita schiaccia i valori, il che rende ogni neurone una sorta di piccolo classificatore (applica ad esempio la SoftMax)

Ora che abbiamo ben chiaro la struttura di una rete neurale, possiamo parlare delle Reti Neurali Ricorrenti, anche dette in inglese Recurrent Neural Network.

Come viene riportato in un articolo in Domsoria, un sito che tratta argomenti di intelligenza artificiale "Le RNN o Recurrent Neural Network sono molto interessanti perchè, a differenza delle feed forward, in cui l'informazione/segnale può andare solo in un verso ed ogni neurone può essere interconnesso con uno o più neuroni della catena successiva, in questo tipo ti reti, i neuroni possono ammettere anche dei loop e/o possono essere interconnessi anche a neuroni di un precedente livello."[5]

Le reti feedforward, che di fatto sono le principali e classiche reti neurali, sono composte quindi da neuroni che in uno stesso livello non possono comunicare tra loro, ma possono inviare segnali solo allo strato successivo. Sono proprio le reti che raffigura la Figura 6.1 precedente.

Al contrario, nelle RNN, possono esistere dei loop, dei cicli, i quali permettono che il segnale venga propagato non solo in avanti, ma anche alla catena precedente. Se andiamo a srotolare il ciclo di una rete neurale ricorrente, otteniamo la seguente Figura 6.3, ripresa dall'articolo di G. Giacaglia, *Come funzionano i trasformatori*, nel sito Towards Data Science.

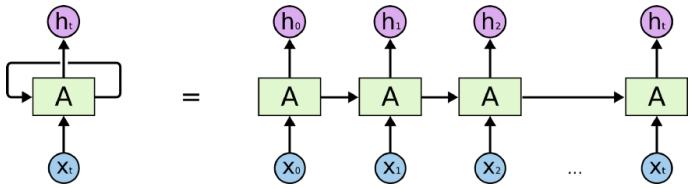

Figura 6.3: Ciclo di una Rete Neurale Ricorrente

Andiamo ad indicare con A la rete neurale che elabora gli input x_i e gli output h_i. Vediamo a destra dell'uguale i vari passaggi che permettono il passaggio delle informazioni elaborate.

Chiaramente possiamo notare che questa elaborazione è correlata a sequenze. Se vogliamo quindi attuare la traduzione di un testo, possiamo impostare ogni input come ogni singola parola del testo che vogliamo tradurre.

[5]Domsoria, *RNN – Come funzionano le Recurrent Neural Network*, `https://www.domsoria.com/2019/11/rnn-recurrent-neural-network/`

Quindi le reti neurali ricorrenti funzionano con dati sequenziali e sono utilizzate, come abbiamo detto, per i problemi di elaborazione del linguaggio naturale; ad esempio per la traduzione della lingua.

In definitiva, questo tipo di rete neurale che utilizza dati sequenziali viene chiamato Sequence to Sequence.
Questi modelli sono costituiti da un codificatore e un decodificatore. L'Encoder prende la sequenza di input e la mappa in un vettore n-dimensionale. Questo vettore poi viene passato al Decoder che lo trasforma in sequenza di output, che in questo caso sarà la traduzione dell'input.

Queste due architetture, codificatore e decodificatore, verranno trattate poi più nello specifico quando parleremo dei Trasformatori.

Esistono diversi tipi di RNN.

- Modelli da vettore a sequenza, i quali prendono un vettore e danno in output una sequenza.
- Modelli da sequenza a vettore, sono utilizzati molto per la sentiment analysis. Come si capisce, hanno per input una sequenza e danno un vettore come output.
- Sequence To Sequnce Models, che prendono una sequenza come input e ne restituiscono un'altra come output. Questi sono i modelli maggiormente utilizzati per attività di traduzione.

Il problema delle Reti Neurali Ricorrenti risiede sulla loro memoria. Prendendo l'esempi pubblicato sempre nell'articolo di G. Giacaglia, andiamo a prendere in esame la frase "le nuvole nel ...", volendo prevedere la parola successiva.

Per il modello è abbastanza ovvio che la parola successiva sarà cielo, ma vediamo che non è sempre così.

Dandogli in input la frase "Sono cresciuto in Francia... parlo fluentemente..." la rete ricorrente non avrà sufficienti informazioni per generare la parola successiva, poichè avremo bisogno del contesto della Francia, che è più indietro nel testo.

Se nel primo caso la differenza tra le informazioni rilevanti e il luogo necessario era abbastanza piccola, nell'ultimo il divario fra le informazioni rilevanti e il punto in cui dobbiamo andare a predire la parola diventa molto più ampio.

Questo perché, dato che abbiamo dei cicli, l'informazione viene trasmessa ad ogni passaggio e più è lunga la catena più è probabile che l' informazione venga persa.

Per cercare di ovviare a questo problema si va ad utilizzare un tipo speciale di RNN, chiamata*Memoria a lungo termine* (LSTM).

Questi modelli hanno la particolarità di avere uno strato in più chiamato *cella di memoria*, conservando informazioni utili sui dati precedenti nella sequenza per aiutare con l'elaborazione di nuovi punti di dati. Ogni cella prende come

input x_i che, nel caso della traduzione di una sequenza, rappresenta una parola, lo stato della cella precedente e l'output della cella precedente. Quello che fa non è altro che un'elaborazione di questi dati per poi generare un nuovo stato della cella e un nuovo output.

In questo modo riusciamo a conservare le informazioni più importanti e anche quelle che abbiamo trattato in un istante di tempo lontano da quello attuale.

Vediamo la struttura di questa memoria a lungo termine nella Figura 6.4 ripresa dal sito di G. Giacaglia.

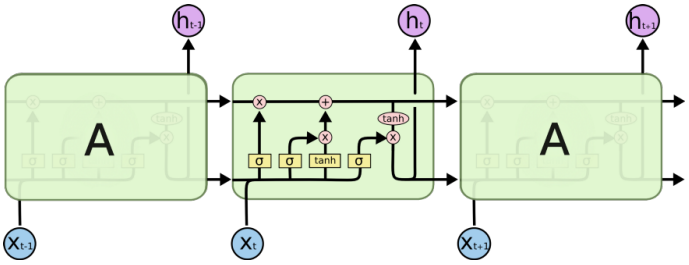

Figura 6.4: Ciclo di una Rete Neurale Ricorrente

Purtroppo però, lo stesso problema che si verifica per le reti neurali ricorrenti, lo ritroviamo con gli LSTM. Anche se per frasi molto più lunghe, non funziona in modo ottimale.

Un altro problema che accomuna entrambi è il fatto che è difficile parallelizzare il lavoro per l'elaborazione delle frasi, in quanto si deve elaborare parola per parola.

Una svolta per la risoluzione di questi problemi sono stati i Trasformatori.

6.2 Trasformatori

Vediamo in dettaglio come funzionano. I Trasformatori sono composti principalmente da un codificatore che opera sulla sequenza di ingresso e da un decodificatore che opera sulla sequenza di uscita e prevede l'elemento successivo nella sequenza.

Un esempio pratico è la traduzione automatica, nella quale il trasformatore può prevedere, data una sequenza di parole, la parola successiva tradotta fino a che a frase non è stata completamente tradotta.

Guardando la Figura 6.5 ripresa da un articolo su ichi.pro[6], andiamo a spiegare i vari passaggi effettuati dal codificatore (a sinistra) e da decodificatore (a destra).

[6]ichi.pro, *Cos'è un trasformatore?*, `https://ichi.pro/it/cos-e-un-trasformatore-229238812437508`

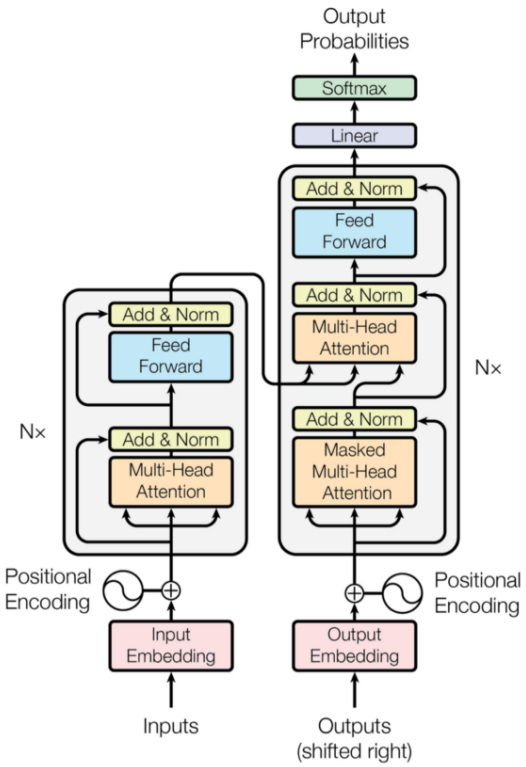

Figure 1: The Transformer - model architecture.

Figura 6.5: Architettura del trasformatore (encoder a sinistra, decoder a destra)

6.3 Encoder

Nell'encoder il primo modulo è *l'incorporamento di input e output*. Inizialmente gli input e gli output vengono incorporati in uno spazio n-dimensionale, poichè come sappiamo non possiamo utilizzare direttamente il testo.

Quindi questo livello di incorporamento prende una sequenza di parole e apprende una rappresentazione vettoriale per ogni parola, in modo da poterla tradurre in un formato comprensibile alla macchina.

Il passaggio successivo, fondamentale per il processo, è *il blocco di codifica posizionale*. Il trasformatore aggiunge ad ogni incorporamento un vettore che rappresenta la posizione di quella parola nella sequenza, poiché, come vedremo successivamente, l'ordine in cui vengono inserite le sequenze è ante. Tutte le parole della sequenza di input vengono inviate senza ordine o posizione speciali, quindi il modello non ha idea di come le parole vengano ordinate.

Matematicamente, i blocchi di codifica posizionale calcolano questo vettore di posizione per ciascuna parola attraverso funzioni seno e coseno. Considerando un input con 10.000 posizioni possibili. Il blocco di codifica posizionale aggiungerà valori di seno e coseno con lunghezze d'onda che aumentano geometricamente da 2Π a $10000 * 2\Pi$.

$$PE(pos, 2i) = \sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$

$$PE(pos, 2i+1) = \cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$

- *pos* è la posizione della parola
- i è la dimensione del vettore
- d è la dimensione di codifica.

Ovvero la dimensione del PE corrisponde a una sinusoidale, con la quale andiamo a codificare il tempo. Utilizziamo il seno per i valori pari e il coseno per i valori dispari.

Vengono quindi concatenati per formare ciascuno dei vettori di codifica posizionale, in modo che il trasformatore possa imparare a riconoscere le differenti posizioni.

Andiamo poi ad effettuare *il meccanismo dell'attenzione*. Il primo passo nel calcolo dell'auto-attenzione consiste nel creare tre vettori da ciascuno dei vettori di input dell'encoder (in questo caso, l'incorporamento di ogni parola).

Quindi per ogni parola creiamo i seguenti vettori.

1. Un vettore di Query Q, ovvero che contiene la rappresentazione vettoriale della parola in questione nella sequenza.
2. Un vettore chiave K, che contiene le chiavi, ovvero le rappresentazioni vettoriali di tutte le parole nella sequenza.
3. Un vettore di valori V, che nel modulo di attenzione è costituito dalla stessa sequenza di parole di Q, ma moltiplicata per i pesi di attenzione di cui parleremo successivamente.

La dimensione di questi vettori, 64, sarà inferiore rispetto al vettore di incorporamento, 512.

Questi vettori vengono creati moltiplicando l'incorporamento per tre matrici che abbiamo addestrato durante il processo di addestramento.

Nella Figura 6.6 ripresa da [6] mostriamo come, moltiplicando i vettori input le matrici di peso W^Q, W^K e W^V, otteniamo i vettori query, chiave e valore associati ad ogni parola della frase.

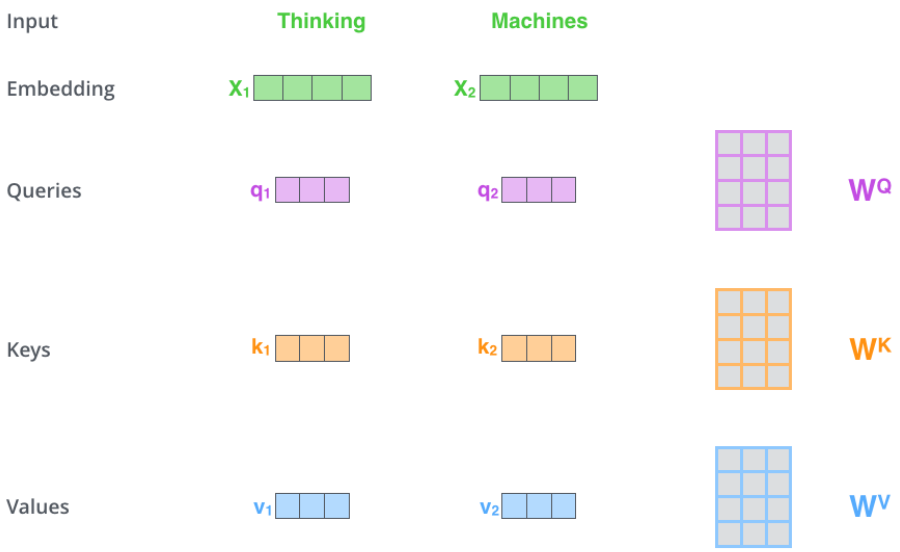

Figura 6.6: Moltiplicazione degli incorporamenti per le matrici di peso

Il secondo passaggio dell'auto-attenzione consiste nel calcolare un punteggio ad ogni parola nella frase di input rispetto alla query.

Questo punteggio determinerà quanta attenzione mettere su determinate parti della frase.

Come facciamo a calcolare questo punteggio? Andiamo a fare il prodotto scalare del vettore query con il vettore chiave della rispettiva parola che stiamo valutando.

Quindi come primo punteggio avremo il prodotto tra q_1 e k_1. Successivamente per q_1 e k_2. Questi punteggi vengono successivamente scalati per la radice quadrata della dimensione delle matrici, quindi per 8 in quanto è la radice quadrata di 64 che indicheremo con n.

Successivamente andiamo ad applicare una funzione Softmax che normalizza i punteggi in modo che siano tutti positivi e con la caratteristica che la loro somma sia uguale a 1. Otteniamo così i valori espressi in probabilità, che sono chiamate "pesi dell'attenzione".

Infine, questi ultimi, vengono moltiplicati per la matrice dei valori e successivamente sommati, ottenendo finalmente l'output finale del processo di attenzione. Possiamo riportare questi passaggi in un unica equazione.

$$Attenzione(Q, K, V) = softmax(\frac{Q*K}{\sqrt{n}}) * V$$

ante è il fatto che le parole di una frase vengono introdotte in input nel trasformatore contemporaneamente, per questo il concetto di ordine, e quindi i blocchi

di codifica posizionale, sono fondamentali.

La vera innovazione dei trasformatori però è stata quella di utilizzare una *Multi-Head Attention*, ovvero un'attenzione multitesta, che consente al modello di prestare attenzione non solo ad un elemento specifico nella frase, come ad esempio può essere il tempo, ma anche a diversi altri punti, in basse al tipo di risultato vogliamo ottenere.

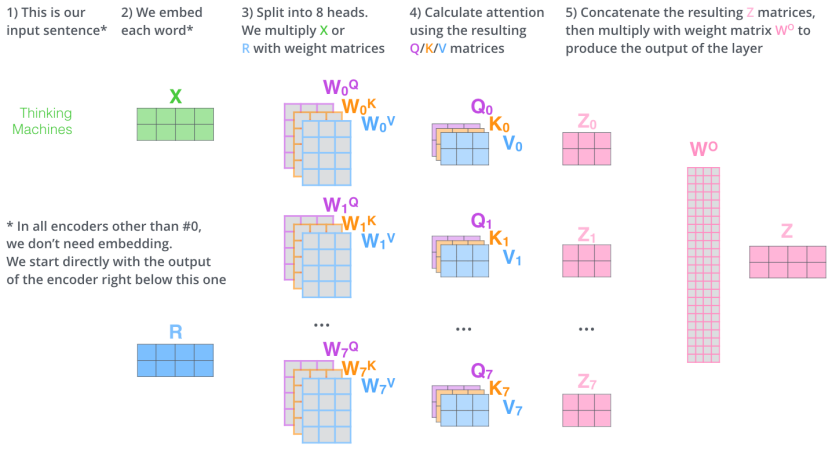

Figura 6.7: Multi-Head Attention

Nell'esempio mostrato da J. Alammar [6] nella Figura 6.7 sopra, manteniamo matrici di peso separate per ciascuna testa di attenzione e le moltiplichiamo per l'input per poi ottenere diverse matrici Q,K, e V per ciascuna.

Se eseguiamo il calcolo ad esempio per 8 Attention Head diverse otterremo altrettante matrici, ma il livello feed-forward, di cui parleremo poi, prevede solamente una singola matrice.

Ciò che facciamo quindi è concatenare le 8 matrici risultanti e moltiplicarle per una matrice di pesi aggiuntiva che indicheremo con W_0. In questo modo otteniamo una matrice Z che rappresenterà il nostro output.

Dopo l'attenzione multitesta, l'output di quest'ultimo viene passato al livello *Add e Norm* il quale somma e normalizza il risultato per poi inviarlo al livello *Feed-Forward*.

Questo strato appena citato è molto semplice, opera sui vettori dell'attenzione e il suo scopo principale è trasformare i vettori di attenzione in una forma accettabile per il successivo codificatore o decodificatore.

In questo caso, dato che stiamo trattando il processo dell'encoder, andremo a passare il risultato al decoder.

6.4 Decoder

Dopo aver terminato la fase di codifica iniziamo quella di decodifica.

Quindi, una volta ottenuto l'output del codificatore, questo risultato viene trasformato in un insieme di vettori di attenzione K e V che verranno utilizzati dal decodificatore per concentrarsi sui punti appropriati della sequenza di input.

Ogni passaggio della fase di decodifica emette un elemento della sequenza di output, come per esempio la traduzione della frase iniziale.

L'output di ogni passaggio viene inviato al decodificatore inferiore, il risultato viene propagato. Incorporiamo e aggiungiamo la codifica posizionale a questi ingressi del decodificatore per indicare la posizione di ogni parola.

Una differenza sostanziale tra lo strato di attenzione nell'encoder e ne decoder è che in quest'ultimo viene utilizzato un blocco di attenzione "mascherato" che permette di leggere solo la parola precedente nella frase tradotta e le altre sono nascoste.

Questo consente al trasformatore di imparare a prevedere la prossima parola tradotta. Queste uscite del blocco di attenzione mascherato vengono successivamente passate al blocco *Add e Norm* per poi essere inviate ad un altro strato di attenzione che riceve al contempo i vettori di attenzione dell'encoder.

Entra poi in gioco uno strato *Feed-Forward* che produce un singolo vettore con una dimensione pari al numero di parole univoche nel vocabolario del modello.

Abbiamo in successione un livello *Lineare* che non fa altro che proiettare il vettore prodotto in un vettore più grande chiamato *logit*, dove ogni cella è una parola del vocabolario.

Infine, con l'applicazione della funzione softmax al vettore, produce un insieme di probabilità corrispondenti a ciascuna parola, che stanno ad indicare la possibilità che una data parola appaia nella traduzione della parla successiva. Quindi viene scelta la cella con la probabilità più alta e la parola associata a questa cella viene prodtta come output successivo per la traduzione.

Come abbiamo detto, questa traduzione avviene in modo iterativo e alla fine del processo viene aggiunto un "marker" per indicare appunto la terminazione e il completamento della traduzione, come possiamo vedere nella Figura 6.8, riportata nell'articolo di J. Alammar [6], che mostra la traduzione della frase francese "Je suis ètudiant" in inglese.

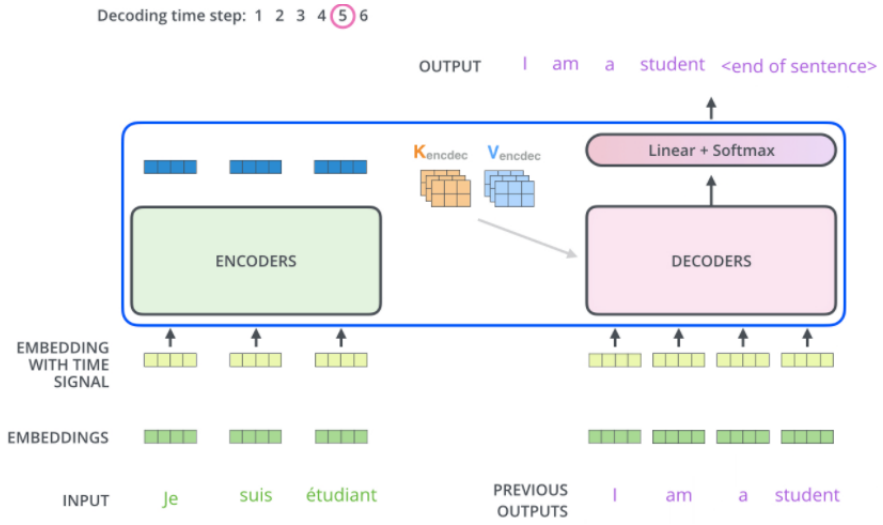

Figura 6.8: Dal codificatore al decodificatore

L'addestramento viene effettuato con la *Perdita di Entropia Incrociata*, confrontando i valori previsti con i valori effettivamente ottenuti.

Il *Natural Language Processing with Transformers*[11], i cui autori sono Lewis Tunstall, Leandro von Werra e Thomas Wolf, è un libro pratico che mostra come addestrare e ridimensionare questi modelli basati sui Transformers utilizzando una libreria di deep learning basata su Python.

Negli anni, sono state introdotte diverse architetture basate sui trasformatori presentati dal documento del 2017 [3]. Quella che a noi ci interessa e che andremo a trattare sarà il modello *BERT*.

6.5 BERT

BERT, acronimo di Bidirectional Encoder Representations from Transformers, è stato presentato nel documento [7] *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, dai ricercatori della Google AI Language.

È un sistema per l'elaborazione del linguaggio naturale, tra cui la traduzione, la risposta alle domande e altre.

Si basa sull'architettura del transformer (rilasciata da Google nel 2017)[3]. Il trasformer, come abbiamo detto, utilizza una rete di encoder e decoder ma, poiché Bert è un modello di pre-training, viene utilizzata solo una rete di encoder, avente lo scopo di apprendere una rappresentazione del testo di input.

La svolta di BERT è che tale modello è addestrato in modo bidirezionale, quindi coglie un significato più profondo del contesto, rispetto ai modelli

linguistici unidirezionali, ovvero che esaminano la sequenza da sinistra verso destra.

L'algoritmo Bert va a processare le parole del testo non una alla volta, ma valutando le relazioni fra ciascuna parola. Ovvero, va a comprendere il significato della frase (anziché della parola) e il contesto della ricerca.

Nel documento citato inizialmente, i ricercatori vanno a descrivere una nuova tecnica che permette l'addestramento bidirezionale dei modelli, che prende il nome di *Masked Language Modeling (MLM)*.

6.5.1 Addestramento

L'addestramento viene fatto in particolare svolgendo contemporaneamente due azioni chiave.

1. Masked Language Modeling,è un sistema che maschera casualmente alcune parole per poterle prevedere correttamente. Quindi possiamo effettivamente prendere una frase incompleta e chiedere all'algoritmo di completarla per noi.

2. Next Sentence Prediction, è una delle principali innovazioni di Bert. Questa garantisce all'algoritmo di apprendere le relazioni tra le frasi. Permette di prevedere la frase successiva di un' altra frase. Ovvero se una ipotesi di frase si adatti logicamente come frase successiva. Questa previsione potrà essere positiva o negativa.

Vediamo come funzionano in dettaglio queste due tecniche.

Nel MLM quello che effettivamente viene effettuato è di mascherare il 15 % delle parole in ciascuna sequenza sostituendole con un token [MASK], prima di inserirle in Bert. Quindi il modello tenta di predire tali parole in base al contesto fornito dalle altre parole non mascherate.

Elenchiamo quello che viene richiesto per effettuare la previsione della parola in uscita.

1. Un livello di classificazione sopra l'output dell'encoder.

2. Moltiplicare i vettori per la matrice di incorporamento, trasformandoli nella dimensione del vocabolario.

3. Calcolare la probabilità di ogni parola nel vocabolario con sfotmax e calcoliamo la perdita di entropia incrociata.

L'output del MLM è rappresentato da vettori di frase in cui i token mascherati sono sostituiti con i token previsti. Ogni token previsto viene quindi sottoposto a softmax su tutte le parole nel vocabolario.

Poi passiamo alla perdita di entropia incrociata rispetto ai valori effettivi, per calcolare la precisione. Questa funzione prende in considerazione solo la previsione dei valori mascherati e ignora la revisione delle parole non mascherate.

Per la previsione della frase successiva (NSP), il modello riceve coppie di frasi A e B come input e da queste impara a prevedere la frase successiva.

Nell'addestramento il 50% delle volte B è la frase successiva effettiva che esegua A, etichettata come IsNext, e il 50% delle volte è una frase casuale del corpus, etichettata come NotNext.

Per distinguere le due frasi nell'addestramento, l'input viene prima elaborato nel seguente modo.

1. Viene inserito un token [CLS] all'inizio della prima frase e un altro token [SEP] alla fine di ogni frase.
2. Viene aggiunta una frase incorporata per ogni token, che indica se si tratta della frase A o della frase B, rispettivamente del primo elemento della coppia o del secondo.
3. Aggiungiamo a ciascun token un elemento posizionale, appunto per indicarne la posizione nella sequenza.

Vediamo le varie elaborazioni nella Figura 6.9, ripresa dall'articolo in Medium di A. Sen [8].

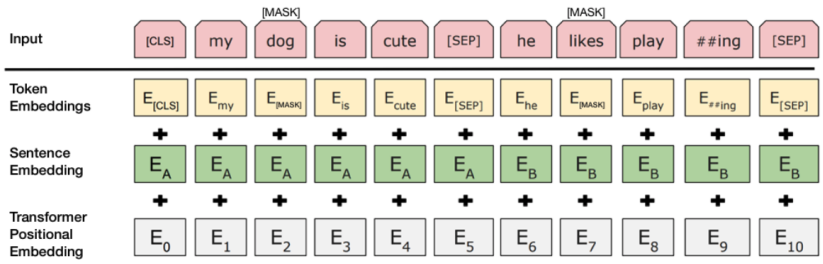

Figura 6.9: Rappresentazione dell'input

Per vedere se effettivamente una frase è collegata alla prima si devono eseguire i seguenti passaggi.

1. L'intera sequenza di ingresso viene sottoposta al modello Transformer.
2. L'output del token [CLS] viene trasformato in un vettore 2x1, utilizzando un livello di classificazione.
3. Andiamo poi a calcolare la probabilità di questo output IsNext, ovvero la probabiità ceh sia la frase successiva, con softmax.

Questi due processi, MLM e NSP, vengono addestrati insieme, con l'obbiettivo di ridurre al minimo la funzione di perdita combinata delle due strategie.

6.5.2 Utilizzo di BERT

Bert, come già ripetuto diverse volte, ha rappresentato senza dubbio una svolta nel campo dell'elaborazione del linguaggio naturale.

Risulta facilmente accessibile grazie a diverse librerie come Keras o tensorflow, e consente un rapido fine-tuning.

Secondo la definizione pubblicata nel sito dell'Università degli studi di Padova "Il fine-tuning è la messa a punto di un sistema affinché questo operi in modo ottimale offrendo le migliori prestazioni."[7] Quindi a seconda della funzionalità che vogliamo ottenere, andremo a ritoccare l'algoritmo opportunamente.

Bert può essere utilizzato per un'ampia varietà di attività linguistiche, aggiungendo solo un piccolo livello al livello principale.

- Possiamo svolgere attività di classificazione, come per esempio la Sentiment Analysis.
  Come abbiamo detto in precedenza, è una tecnica che si occupa di comprendere lo stato d'animo con cui una frase viene espressa o scritta.
  Queste attività vengono svolte in modo simile alla classificazione della frase successiva, aggiungendo un livello di classificazione sopra l'output del trasformatore per il token [CLS]. (Figura 6.10 (b))

- Viene utilizzato per attività di risposta alle domande, ad esempio SQuAD.
  La definizione di SQuAD, acronimo di Standford Question Answering Dataset, è riportata nel sito ufficiale SQuAD: "è un dataset di comprensione della lettura, costituito da domande poste dai crowdworker su una serie di articoli di Wikipedia, in cui la risposta a ogni domanda è un segmento di testo, o span, dal corrispondente passaggio di lettura, o la domanda potrebbe essere senza risposta."[8]
  Il software riceve una domanda relativa ad una sequenza di testo ed elabora la risposta nella sequenza.
  Utilizzando BERT, è possibile addestrare un modello di domande e risposte apprendendo due vettori aggiuntivi che segnano l'inizio e la fine della risposta. (Figura 6.10 (c))

- possiamo utilizzare BERT anche per la Named Entity Recognition (NER), che , come abbiamo detto in precedenza, è un software che riceve una sequenza di testo e va a contrassegnare i veri tipi di entità che compaiono nel testo, come persona, mese, data ecc. Il modello BERT può essere

[7] Università degli Studi di Padova, *Tuning*, `https://www.ict.unipd.it/attivita/infrastruttura-e-piattaforme/tuning-e-problem-determination/tuning`

[8] SQuAD,*SQUAD 2.0 The Stanford Question Answering Dataset*,`https://rajpurkar.github.io/SQuAD-explorer/`

utilizzato per addestrare un modello NER aggiungendo un livello di classificazione che alimenta il vettore output di ciascun token con etichette NER. (Figura 6.10 (d))

Possiamo vedere le varie modellazioni nella Figura 6.10 ripresa dall'articolo di A. Sen [8]. La figura *a* rappresenta il modello BERT nella classificazione della frase successiva, la figura *b* rappresenta il caso di Sentiment Analysis, la figura *c* la Question answering e la *d* il modello Named Entity Recognition.

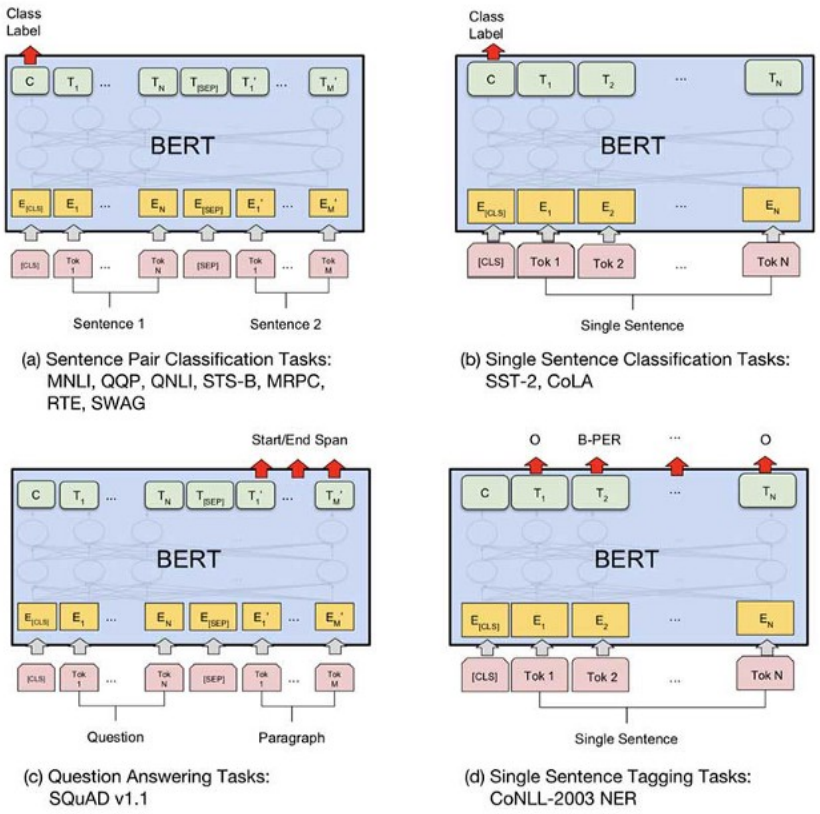

Figura 6.10: BERT fine-tuning

Nella messa a punto di BERT in questi modelli, la maggior parte degli iperparametri rimane la stessa, ciò significa che le modifiche sono lievi.

Gli iperparametri sono quelle proprietà che sono impostate in anticipo e non possono essere cambiate durante l'allenamento.

Il modello BERT è stato fondamentale soprattutto per le query di ricerca, diventando parte integrante dei motori di ricerca di Google.

In particolare nel 2019, l'azienda ha rilasciato negli Stati Uniti (in inglese) un aggiornamento riguardante l'algoritmo Google BERT [9]. Quest'ultimo consente di interpretare meglio l'intera frase per dare risultati più accurati, invece che elaborare il testo di ricerca prola per parola come facevano i modelli precedenti. Con questo nuovo aggiornamento Google sta migliorando la capacità di interpretare ciò che gli utenti stanno veramente cercando quando interrogano il suo motore.

Il primo esempio, pubblicato nell'aggiornamento del 2019 [9], riguardava i risultati della ricerca della query "2019 brazil traveler to usa need a visa.".

In precedenza all'algoritmo BERT, i motori di ricerca non capivano l'importanza della relazione delle parole con la parola "to" e questo restituiva risultati inerenti ai cittadini statunitensi che si recavano in Brasile, che è ovviamente errato. Dopo l'aggiornamento con BERT, la ricerca è in grado di cogliere questa sfumatura e l'importanza della parola "to", restituendo risultati più pertinenti.

Come mostra la Figura 6.11, pubblicata sull'articolo dell'aggiornamento della ricerca di Google con BERT [9], vediamo i risultati prima e dopo la modifica dell'algoritmo.

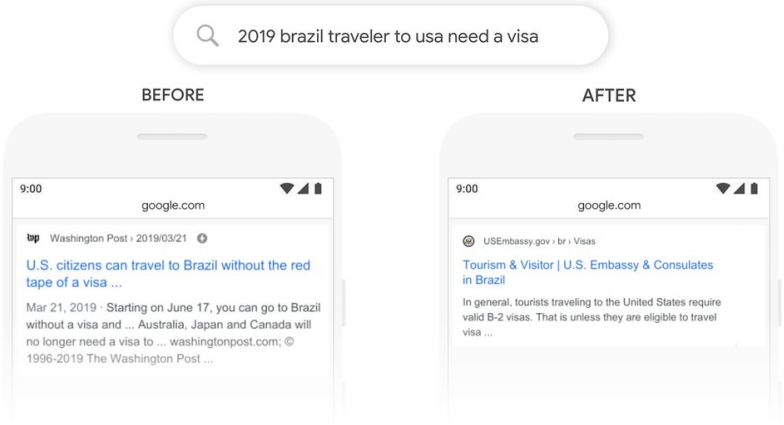

Figura 6.11: Esempio miglioramento ricerca con Google BERT

6.6 Vantaggi e Svantaggi

In definitiva, come viene riportato nell'articolo *Bert: Il nuovo aggiornamento Google spiegato in modo semplice* sul sito Imprendo[9] "il nuovo aggiornamento

[9]Imprendo, *Bert: Il nuovo aggiornamento Google spiegato in modo semplice*, `https://imprendo.blog/2019/11/05/bert-il-nuovo-aggiornamento-google-spiegato-in-modo-semplice/`

Google BERT potrebbe essere uno dei più significativi degli ultimi 5 anni per quanto riguarda il SEO – ottimizzazione dei contenuti per la migliore resa nel motore di ricerca."

Ma non solo. Come abbiamo già specificato, BERT di pone anche come soluzione ai precedenti algoritmi per le tecniche di elaborazione del linguaggio naturale, soppiantando i problemi che Word2Vec non riusciva a risolvere.

Quindi possiamo elencare di seguito i vantaggi che tale algoritmo comporta.

- Miglioramento dell'attività di SEO e della comprensione del linguaggio, nonché del contesto del testo della ricerca, dando valore alle preposizioni.

Risoluzione dei problemi di Word2Vec.

- Differenzia le parole con uguali sintassi, ma diverso significato, riuscendo ad elaborare in modo migliore il contesto di cui fanno parte. Prendiamo l'esempio in Figura presente nell'articolo *Demystifying BERT: A Comprehensive Guide to the Groundbreaking NLP Framework* del computer science Mohd Sanad Zaki Rizvi pubblicato su sito Analytics Vidhya[10].

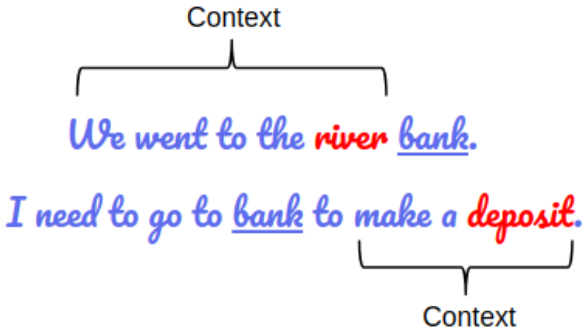

Figura 6.12: La stessa parola "bank" utilizzata in diversi contesti

Vediamo che la parola "bank" nelle due frasi prende un significato diverso rispetto alla frase.

BERT coglie questa sfumatura e riesce a distinguere che sono due parole distinte, al contrario di Word2Vec che andrà a generare lo stesso vettore per entrambe.

- Si terrà conto della posizione delle parole nella frase, al contrario di Word2Vec.

[10]Mohd Sanad Zaki Rizvi, *Demystifying BERT: A Comprehensive Guide to the Groundbreaking NLP Framework*, Analytics Vidhya, `https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/`

– Supporta *l'Out Of Vocabulary* (OOV), ovvero le parole fuori dal vocabolario. Il modello riesce a generare vettori di incorporamento per parole che non risiedono già nel vocabolario.

Il più grande svantaggio dell'utilizzo di BERT però sono le risorse computazionali necessarie per addestrare e fare inferenze.

"BERT è una tecnologia per generare word embedding/vettori "contestualizzati", che è il suo più grande vantaggio, ma anche il più grande svantaggio in quanto è molto impegnativo in termini di calcolo in fase di inferenza , il che significa che se si desidera utilizzarlo in produzione su larga scala, può diventare costoso", come viene riportato nell'articolo di Stephen Jeske, *Google BERT Update and What You Should Know*, nel sito Market Muse[11].

[11] Stephen Jeske, *Google BERT Update and What You Should Know*, Market Muse, `https://blog.marketmuse.com/google-bert-update/`

Conclusioni

In questo elaborato abbiamo visto l'importanza del Natural Laguage Processing, e quindi dell'elaborazione del testo, unito ai modelli di apprendimento automatico.

Il linguaggio umano è sorprendentemente complesso e diversificato. Ci esprimiamo in infiniti modi, sia verbalmente che per iscritto.

Esistono centinaia di lingue e dialetti, il che rende complicato elaborare sistemi capaci di accomunare tutte queste diversità.

L'elaborazione del linguaggio naturale aiuta i computer a comunicare con gli esseri umani nella propria lingua e a consentire un interazione proficua tra umano e macchina.

La NLP consente ai computer di leggere il testo, ascoltare il parlato, interpretarlo, misurare il sentimento e determinare quali parti sono importanti.

Gli algoritmi di apprendimento automatico, che sono strumenti base per la NLP, codificano il linguaggio umano in linguaggio macchina, garantendo appunto questa interazione tra uomo e computer.

Tutto ciò ci permette di creare sistemi intelligenti capaci di emulare la logica e le azioni umane.

Riuscendo a sviluppare sistemi in grado di apprendere in modo autonomo, ovvero basati su un apprendimento supervisionato, da un enorme quantità di dati, siamo riusciti a fare enormi passi avanti nella scienza.

Basta pensare alla guida autonoma o anche ai moderni sistemi di assistenti intelligenti che di recente hanno preso una grande fetta di mercato, fornendo agli utenti la possibilità di interagire tramite comandi vocali.

Siamo partiti nel Capitolo 1 con una breve storia sull'elaborazione del linguaggio naturale e dello sviluppo nel tempo dei vari modelli di apprendimento automatico, ponendo attenzione sui collegamenti tra questi, i punti di forza, i vari problemi e sulle loro funzionalità pratiche.

Nel Capitolo 2 abbiamo trattato i modelli di apprendimento in base alla Classificazione, ovvero quel processo di analisi delle sequenze di testo etichettato, definendo le relazioni tra i dati e le loro classi di appartenenza.

In pratica serve ad esempio per i sistemi di riconoscimento di email spam o anche per riconoscere il sentiment di una recensione.

La classificazione del testo è alla base di quasi tutte le attività di intelligenza artificiale o apprendimento automatico che coinvolgono l' elaborazione del linguaggio naturale.

Poi siamo passati a presentare una leggera introduzione sul metodo di apprendimento supervisionato, fondamentali per parlare dei modelli di apprendimento automatico.

Successivamente ci siamo concentrati sulle tecniche del NLP, ovvero sulle fasi di etichettamento dei dati, preprocessing e trasformazione ed elaborazione del testo. Tutte queste appena elencate ci permettono di andare a semplificare e andare a ridurre "rumore" al testo, per facilitare la sua elaborazione negli algoritmi di apprendimento.

Con la trasformazione del testo abbiamo poi iniziato ad introdurre i primi algoritmi, partendo da Bag of Words, trattato nel Capitolo 3.

Quest'ultimo, come abbiamo visto, si concentra sulle occorrenze della parola, ma, anche se si tratta di un algoritmo semplice e facile da implementare, presentava alcune problematiche, in quanto l'elaborazione del testo rimaneva superficiale e poco accurata. In particolare l'ordine delle parole veniva ignorato e non apportava alcuna informazione sul significato del testo.

Ad esempio considerando due frasi del tipo: "La lezione è noiosa, ma facile" e "La lezione è facile, ma noiosa", tale modello svilupperebbe gli stessi vettori per entrambe le frasi, non apportando nessun significato importante.

Nel Capitolo 4 introduciamo il modello TF-IDF, che di fatto è un'implementazione del Bag of Words la quale tenta di sopprimere i problemi di quest'ultimo.

La novità che TF-IDF presentava era che, al posto di contare le occorrenze delle parole come faceva l'altro, andava a valutare la frequenza con cui le parole apparivano nel documento.

In questo modo si veniva data importanza alle parole che apparivano comunemente nel nostro testo, svantaggiando quelle che comparivano di meno.

Questo aiutava a catturare le parole chiave di un documento. In pratica tale algoritmo viene utilizzato soprattutto nella SEO, per ottimizzare i contenuti della pagina tramite le parole chiave e posizionare quel sito ad un altro livello nei risultati del motore di ricerca.

Questo algoritmo però non ci diceva nulla sul significato contestuale delle parole, poiché si basava solo sulla loro frequenza.

Per questo, nel Capitolo 5, abbiamo definito i modelli Word2Vec, introducendo anche la definizione dei Word Embedding, incorporamenti di parole, fondamentali nella storia dell'apprendimento automatico.

Abbiamo spiegato la differenza tra il modello Skip Gram e CBOW, il loro processo di apprendimento tramite Forward propagation e la loro fase di ottimizzazione con la Back Propagation, esponendo infine i vari vantaggi e svantaggi.

L'algoritmo Word2Vec era in grado di trasformare le parole rappresentandole in uno spazio vettoriale, in questo modo era possibile rappresentare le loro somiglianze semantiche dalla vicinanza e la lontananza delle parole. Diciamo che parole simili producevano vettori simili.

Con questo processo si riusciva a cogliere efficacemente le relazioni tra le parole attraverso il loro contesto, a differenza dei modelli precedenti.

Un altro vantaggio era la capacità di apprendere word embedding di qualità e con poca complessità di spazio e tempo. Era computazionalmente più efficiente rispetto ai precedenti.

Presentava però alcuni lati negativi, in quanto l'approccio con Word2Vec non era ottimale.

Gli incorporamenti che andava a generare non erano legati in alcun modo con il contesto a cui appartenevano. Ciò significava che parole uguali sintatticamente, ma diverse semanticamente erano rappresentate allo stesso modo. In più l'ordine delle parole non era mantenuto.

Lo svantaggio più grande che presentava era però il fatto che non supportava le parole fuori dal vocabolario, in quanto non riusciva a elaborare parole che non risiedevano nel suo vocabolario. Questo viene chiamato *Out of Vocabulary*.

Con i più recenti modelli, i Transformer, che abbiamo trattato nel Capitolo 6, si vanno a risolvere tutti i problemi che Word2Vec presentava, dando una svolta radicale al mondo del Natural Language Processing.

In questo capitolo abbiamo introdotto la definizione e la struttura di un trasformatore. Abbiamo posto particolare attenzione ad un tipo di modello Transformers, conosciuto come BERT. Una volta definito quest'utlimo abbiamo parlato del suo addestramento e del suo utilizzo pratico.

Infine abbiamo spiegato i miglioramenti apportati rispetto a Word2Vec. In particolare risolveva tutti i lati negativi che quest'ultimo presentava, come la differenziazione di parole sintatticamente uguali, ma semanticamente diverse.

Risolveva l'*Out of Vocabulary*, potendo generare vettori di incorporamento per parole anche che non erano presenti nel suo vocabolario. Abbiamo anche visto come teneva conto della posizione delle parole nella frase di input.

La svolta più significativa che si ebbe con i trasformatori però fu con il meccanismo dell'*Attention*, utilizzato per determinare su quali parti del testo concentrarsi e dare più peso. Grazie a questo i trasformatori sono riusciti a superare le prestazioni delle Reti Neurali Ricorrenti, che fino a quel momento erano le sovrane nell'analizzare in modo sequenziale i dati di testo per la previsione.

Il lato negativo però risiede sulle risorse di calcolo necessarie nella fase di addestramento e di inferenza.

Anche se i trasformatori sono molto impegnativi in termini di calcolo, rappresentano comunque gli attuali modelli di NLP per analizzare i dati di testo.

Nonostante i risultati ottimi ottenuti in differenti campi d'applicazione, l'esigenza di migliorare ulteriormente le capacità di comprensione automatica del linguaggio naturale per avvicinarsi il più possibile alla vera e propria intelligenza umana, rappresenta tuttora una sfida su cui il mondo della ricerca sta lavorando.

Si hanno continui sviluppi in questa materia. Un delle ultime innovazioni nell'ambito del Natural Language Processing è il GPT-3 di Open AI, basato su Transformers, con il quale si è registrato un traguardo importante nel mondo dell'interpretazione del linguaggio naturale. Come scrivono i ricercatori del Politecnico di Milano nel loro sito *Osservatori.net* "OpenAI, organizzazione no profit per la ricerca sull'intelligenza artificiale, ha rilasciato il suo ultimo modello linguistico basato su reti neurali – GPT-3 o Generative Pre-trained Transformer 3 –, che a oggi risulta la rete con più parametri mai addestrata."[12]

Questo modello ha numerose potenzialità. Può rispondere a domande aperte, creare poesie e storie da zero e addirittura programmare pagine web.

Ma GPT-3 non è l'ultimissima innovazione in campo NLP. Durante il Google I/O 2021 è avvenuta la presentazione di LaMDA: una tecnologia basata sulla stessa architettura, ma con un'ulteriore slancio. Come è riportato nell'articolo di Alessio Pomaro nel suo blog "LaMDA, acronimo di Language Model for Dialogue Applications, è un potente algoritmo di Google per la comprensione del linguaggio naturale."[13]

Questo sistema presentò una notevole capacità di intrattenere conversazioni naturali tra l'intelligenza artificiale e l'umano.

La caratteristica è che genera le frasi in base alle informazioni che gli vengono fornite. È basato anche questo modello sui Transformers e riesce a cogliere le relazioni tra le parole predicendo quali tra queste sono statisticamente più indicate per proseguire. A differenza tra gli altri modelli è che è una tecnologia addestrata principalmente al dialogo. Ancora questo modello non è attualmente in produzione, ma è ancora in fase di sviluppo per poi essere utilizzato, una volta completato, in prodotti come assistenti vocali, ricerche ecc.

Come abbiamo già detto, l'elaborazione del linguaggio naturale porta con sé molte difficoltà, date dalla complessità e dall'ambiguità della lingua.

È fondamentale saper riconoscere i diversi significati che può assumere una parola, il tono della voce che può cambiare il senso di una frase e soprattutto il contesto e l'ambiente che rappresenta quel testo.

Importante è anche avere a disposizione modelli per l'elaborazione del testo efficienti che riescano appunto a cogliere tali ambiguità attraverso l'apprendimento automatico, che è alla base dell'intelligenza artificiale.

[12] Osservatori.net, Natural Language Processing (NLP): come funziona l'elaborazione del linguaggio naturale, `https://blog.osservatori.net/it_it/natural-language-processing-nlp-come-funziona-lelaborazione-del-linguaggio-naturale`

[13] A.Pomaro, Cos'è LaMDA? La comprensione del linguaggio naturale secondo Google *https://www.alessiopomaro.it/lamda-linguaggio-naturale-google/*

Ovvero creare algoritmi capaci di modellarsi e prendere forma attraverso una grande quantità di dati che gli vengono forniti in input, in modo da imparare e svolgere svariati compiti in modo autonomo.

I progressi in questo campo sono evidenti e non accennano a rallentare. Ci sarà uno sviluppo tecnologico in molte aree. Basti pensare ai recenti droni, veivoli radiocomandati che vengono impiegati in molti task.

Lo sviluppo dei droni ha fatto passi da gigante, partendo dalle riprese fotografiche, scopi di intrattenimento ecc.. fino ad arrivare addirittura ad essere utilizzati in ambito agricolo, utilizzati per la somministrazione di fertilizzanti o insetticidi. In campo civile, tramite monitoraggio del territorio.

Un impiego importante riguarda l'ambito ambientale, in quanto vengono utilizzati per rilevare la presenza di agenti contaminanti sia in aria che in acqua. Sono in grado addirittura di rilevare la presenza di discariche abusive.

L'obbiettivo di questi sistemi però non si ferma qui. Molte sono le applicazioni future che si prevedono attuabili.

Sono attualmente in fase di sviluppo i cosiddetti "droni pompiere" che, come si capisce dal termine, interverranno in caso di incendi, essendo in grado di spegnere incendi e monitorare l'area verificando lo stato dell'incendio.

Un altro importante ambito che si andrà a ricoprire sarà di tipo logistico. Si prevede l'impiego di droni per effettuare consegne veloci ed affidabili. Non ci sarà così più bisogni di corrieri per trasportare merci.

Per questo ultimo punto è bene parlare di due grandi filoni di pensiero. La visione utopica, che favorisce e punta allo sviluppo di sistemi intelligenti per creare una nuova era.

La visione apocalittica, che al contrario ha sfiducia nello sviluppo tecnologico visto negativamente, in quanto andrà a sostituire molti lavori umani.

Indipendentemente da quale visione si ritenga più realistica, l'avanzata della tecnologia e di sistemi intelligenti è sempre più veloce, apportando nuovi cambiamenti ogni giorno.

È evidente che la scienza e il progresso dell'elaborazione del linguaggio naturale è fondamentale e sempre più persistente.

L'elaborazione dei dati diventa ogni giorno più accurata e precisa, riuscendo a cogliere le varie ambiguità del linguaggio parlato e scritto, per creare macchine sempre più vicine all'intelligenza umana, capaci di prendere decisioni autonome.

La velocità con cui nuove tecniche e nuovi modelli prendono forma è impressionante e ho ragione di credere che è su questi sistemi di intelligenza artificiale che si farà largo un nuovo mondo, non più troppo lontano, il quale rivoluzionerà completamente la vita così come la vediamo.

Bibliografia

[1] W. Weaver, (1949), *Translation.*

[2] T. Mikolov, I. Sutskever, K. Chen, G. Corrado e J. Dean, (2013), *Distributed representations of words and phrases and their compositionality.*

[3] A.Vaswani, N.Shazeer, N.Parmar, J.Uszkoreit, L.Jones, A.Gomez, L.Kaiser e I. Polosukhin, (2017), *Attention is all you need.*

[4] D. Carlini, *Classificazione Testuale attraverso il Machine Learning*, in Tech Blog, (2020), `https://techblog.smc.it/it/2020-12-21/nlp-text-clf`

[5] A. Sen, *Text Classification, From Bag-of-Words to BERT, Part 2 (Word2Vec)*, in Medium, (2020), `https://medium.com/analytics-vidhya/text-classification-from-bag-of-words-to-bert-part-2-word2vec-35c8c3b34ee3`

[6] J. Alammar, The Illustrated Transformer (2018), `http://jalammar.github.io/illustrated-transformer/`

[7] J. Devlin, M.W. Chang, K. Lee e K. Toutanova, (2018), *Bert: Pre-training of deep bidirectional transformers for language understanding*, `https://arxiv.org/pdf/1810.04805.pdf`.

[8] A. Sen, *Text Classification — From Bag-of-Words to BERT — Part 6(BERT)*, `https://medium.com/analytics-vidhya/text-classification-from-bag-of-words-to-bert-part-6-bert-2c3a5821ed16`

[9] Pandu Nayak, *Understanding searches better than ever before*, `https://blog.google/products/search/search-language-understanding-bert/`.

[10] Ankur A. Patel, Ajay Uppili Arasanipalai, Applied Natural Language Processing in the Enterprise, `https://www.oreilly.com/library/view/applied-natural-language/9781492062561/`

[11] Lewis Tunstall, Leandro von Werra, Thomas Wolf, Natural Language Processing with Transformers, `https://www.oreilly.com/library/view/natural-language-processing/9781098103231/`

[12] Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta, Harshit Surana, Practical Natural Language Processing, https://www.oreilly.com/library/view/practical-natural-language/9781492054047/

[13] Ian Goodfellow and Yoshua Bengio and Aaron Courville, Deep Learning An MIT Press book, https://www.deeplearningbook.org/

Ringraziamenti

Desidero ringraziare chi mi è stato vicino e mi ha accompagnato in questo percorso di studio.

Vorrei ringraziare il mio relatore Francesco Santini per la sua disponibilità e per avermi guidato durante il periodo di tirocinio e nella stesura di questa tesi.

Un ringraziamento speciale lo dedico a mia madre e a mio fratello che mi hanno sempre supportato e sopportato in questo percorso, soprattutto negli ultimi mesi di alti e bassi.

Infine, al mio collega Giulio per avermi sostenuto e aver creduto in me.