ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria
Corso di Laurea in Ingegneria e Scienze Informatiche

VERBALIZZAZIONE DI EVENTI BIOMEDICI ESPRESSI NELLA LETTERATURA SCIENTIFICA: GENERAZIONE CONTROLLATA DI LINGUAGGIO NATURALE DA GRAFI DI CONOSCENZA MEDIANTE TRANSFORMER TEXT-TO-TEXT

Elaborato in
Programmazione di applicazioni data intensive

Relatore
Prof. Gianluca Moro

Co-relatore
Dott. Giacomo Frisoni

Presentata da
Lorenzo Balzani

Seconda Sessione di Laurea
Anno Accademico 2020 – 2021

PAROLE CHIAVE

Natural Language Processing

Graph verbalization

Pre-trained Language Models

Event Extraction

Biomedical literature

I tre elementi essenziali per ottenere qualsiasi cosa valga la pena avere sono; primo, duro lavoro, secondo, perseveranza, e terzo, buonsenso.

Thomas Edison

Abstract

Il periodo in cui viviamo rappresenta la cuspide di una forte e rapida evoluzione nella comprensione del linguaggio naturale, raggiuntasi prevalentemente grazie allo sviluppo di modelli neurali. Nell'ambito dell'information extraction, tali progressi hanno recentemente consentito di riconoscere efficacemente relazioni semantiche complesse tra entità menzionate nel testo, quali proteine, sintomi e farmaci. Tale task – reso possibile dalla modellazione ad eventi – è fondamentale in biomedicina, dove la crescita esponenziale del numero di pubblicazioni scientifiche accresce ulteriormente il bisogno di sistemi per l'estrazione automatica delle interazioni racchiuse nei documenti testuali. La combinazione di AI simbolica e sub-simbolica può consentire l'introduzione di conoscenza strutturata nota all'interno di language model, rendendo quest'ultimi più robusti, fattuali e interpretabili. In tale contesto, la verbalizzazione di grafi è uno dei task su cui si riversano maggiori aspettative. Nonostante l'importanza di tali contributi (dallo sviluppo di chatbot alla formulazione di nuove ipotesi di ricerca), ad oggi, risultano assenti contributi capaci di verbalizzare gli eventi biomedici espressi in letteratura, apprendendo il legame tra le interazioni espresse in forma a grafo e la loro controparte testuale. La tesi propone il primo dataset altamente comprensivo su coppie evento-testo, includendo diverse sotto-aree biomediche, quali malattie infettive, ricerca oncologica e biologia molecolare. Il dataset introdotto viene usato come base per l'addestramento di modelli generativi allo stato dell'arte sul task di verbalizzazione, adottando un approccio text-to-text e illustrando una tecnica formale per la codifica di grafi evento mediante testo aumentato. Infine, si dimostra la validità degli eventi per il miglioramento delle capacità di comprensione dei modelli neurali su altri task NLP, focalizzandosi su single-document summarization e multi-task learning.

Introduzione

Negli ultimi anni, i *language models* (*LM*) hanno raggiunto incredibili performance, superando quelle umane in alcuni task legati alla comprensione del testo. Tra i modelli generativi più usati e famosi vi sono T5 [1] e BART [2]. Tali risultati sono stati resi possibili dalla tendenza ad accrescere costantemente le dimensioni dei modelli di *Natural Language Processing* (*NLP*)/*Natural Language Understanding* (*NLU*) in termini architetturali. Trend esponenziale manifestatosi negli ultimi anni e reso evidente con i recenti rilasci di *OpenAI GPT-3* [3] (175B) e *GShard* [4] (600B). In Figura 1 una panoramica delle dimensioni dei *LM* e dell'andamento di crescita delle loro architetture.

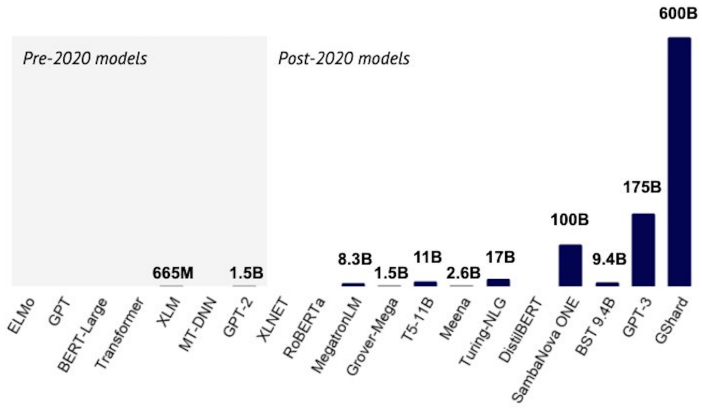


Figura 1: Trend di crescita dei *LM*.
Fonte: `https://www.stateof.ai/`

Tale aumento di dimensioni porta a una diminuzione della democratizzazione con cui viene portata avanti la ricerca: solo alcuni colossi come Google o OpenAI, che posseggono immani risorse computazionali ed economiche, possono sostenere l'addestramento di *LM* con tali dimensioni architetturali. Nonostante ciò, alcuni team di ricerca sono riusciti a competere con l'attuale *state-of-the-art* (*SOTA*), come ad esempio in [5], dove sono state superate le performance di *GPT-3* usando modelli anche 430 volte più contenuti. Nonostante le loro performance, i recenti lavori di ricerca hanno messo in luce la natura fragile di tali sistemi

(attacchi avversari, e.g., [6]) e la loro frequente produzione di testo non fattuale (i.e., allucinazioni), come nell'esempio in Figura 2.

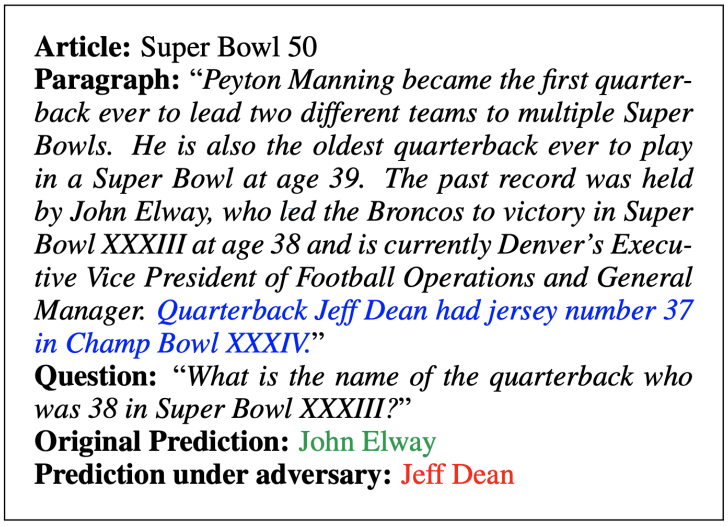


Figura 2: Un esempio di allucinazione in un task di *Question Answering* (*QA*). Il modello originariamente predice la **risposta giusta**, ma è successivamente ingannato dall'aggiunta di un'**ulteriore frase**, producendo un'**allucinazione**.
Fonte: [6]

La produzione di testo non fattuale è legato a una non reale "comprensione" del testo. In tal senso, il recente rilascio di GPT-3 ha acceso una forte discussione all'interno della comunità di linguistica computazionale, portando a definire gli odierni *LM* come dei "pappagalli stocastici" [7, 8]. In un simile scenario, sorge il problema dell'affidabilità di queste reti, fondamentale in campi applicativi quali dominio biomedico e legislativo. Il bisogno di verificare la conoscenza appresa dai modelli ha recentemente portato alla nascita di una nuova area di ricerca denominata *XAI* [9] (Explainable Artificial Intelligence). Infatti, le attuali soluzioni di deep learning sono spesso considerate delle black-box, dove la conoscenza – immagazzinata in matrici di pesi a dimensionalità sempre più elevata – non è direttamente interpretabile da esseri umani. Ad esempio, senza avere la completa comprensione di come si comporti una rete, un veicolo a guida autonoma potrebbe investire un pedone, una diagnosi di malattia potrebbe essere erroneamente rilasciata a una persona sana (o viceversa), oppure un giudice AI estone potrebbe ingiustamente condannare un innocente. In conclusione, a causa della natura sempre meno interpretabile dei *LM*, l'*AI* subsimbolica rischia di diventare progressivamente meno controllabile e affidabile. La comunità scientifica è quasi unanimemente concorde sul fatto che in un futuro prossimo la regolazione di software facente uso di *AI* porterà

a diminuire la presenza di sistemi aventi un comportamento non spiegabile o dotati di conoscenza non verificabile. Soprattutto in ambiti dove decisioni sbagliate possono compromettere la salute o i diritti dell'uomo, la ricerca di modelli "spiegabili" è ritenuta essenziale.

Per far fronte a queste esigenze, molti ricercatori stanno notoriamente promuovendo la combinazione di *AI* subsimbolica e simbolica (e.g., Web Semantico), mirata a fondere i punti di forza di entrambe e a risolvere le debolezze reciproche. La prima, infatti, è in grado di estrarre automaticamente fatti relazionali espressi nel testo per popolare dinamicamente *knowledge graph* (*KG*) di elevata dimensione, altrimenti non sostenibile con annotatori umani e/o esperti di dominio. La seconda, invece, può fornire strumenti di ricerca fondamentali per la risoluzione delle problematiche attuali del deep learning, quali risorse simboliche iniettabili all'interno delle reti neurali o rappresentazioni di conoscenza interpretabili, su cui inferire conoscenza addizionale mediante processi di reasoning logico. Sotto quest'ultimo punto di vista, il linguaggio naturale ha bisogno di un nuovo paradigma e una possibile soluzione potrebbe essere costituita da un intervento alla base: evitare che un *LM* sia chiamato ad apprendere ogni volta conoscenza da zero, ma trarre vantaggio dall'elevato quantitativo di conoscenza linguistica (es. FrameNet), di mondo (es. WikiData) e di dominio già nota ed espressa in modo strutturato. Inoltre, tali rappresentazioni strutturate - principalmente in forma a grafo - sono interpretabili, a differenza di matrici di pesi. Questa direzione di ricerca, focalizzata sulla combinazione di *LM* e conoscenza relazionale, ha consentito ai paper degli ultimi anni di generare testo più fattuale [10] e migliorare le performance su task *NLP* complessi quali open-domain QA [11].
Come possono essere iniettati grafi dentro *LM*? Le soluzoni proposte nel tempo sono molteplici:

1. sequenzializzare i grafi mediante algoritmi di cammino;
2. usare *graph neural network* (*GNN*), divenute popolari dal 2018 per applicare DL direttamente su dati a grafo;
3. verbalizzazione di grafi (un approccio recentemente sperimentato da Google per estendere dataset di pre-training con documenti artificiali esprimenti conoscenza logica di alto livello e valore).

La letteratura, specie quella biomedica, cresce di anno in anno a velocità esponenziale. Monitorare e consultare dettagliatamente pubblicazioni d'interesse è ormai divenuto altamente dispendioso. Sulla base dei dati riportati nel 2020, ogni minuto vengono registrati su *PubMed* oltre 3 paper[1] [12]. Sorge

[1]Risultati bibliometrici ottenuti eseguendo la query "1700:2021[dp]" sul motore di ricerca di PubMed, `https://www.ncbi.nlm.nih.gov/pubmed/`

pertanto la necessità di sistemi automatici per l'esplorazione della conoscenza espressa in letteratura. Attualmente, in ambito biomedico, le rappresentazioni più sofisticate per la modellazione semantica di fatti e interazioni espresse nel testo sono gli eventi. Quest'ultimi – formalizzabili come grafi – consentono di rappresentare in modo strutturato il principale contenuto informativo racchiuso nelle pubblicazioni scientifiche. Sinora il principale task trattato legato agli eventi è la loro estrazione dal testo. Risultano totalmente assenti paper che affrontino il task "contrario", ovvero la verbalizzazione di eventi, nonostante esso sia fondamentale per numerose applicazioni ad alto impatto. Oltre a quelle già descritte, si aggiungono:

- la spiegazione tramite linguaggio naturale di conoscenza simbolica;
- lo sviluppo di una *chatbot* più efficace per QA biomedicale, con domande atte a verificare, ad esempio, la presenza o meno di un'interazione "causa" o "cura" tra un medicinale e una patologia. Se non sfruttassimo conoscenza strutturata come gli eventi, si potrebbero produrre con più facilità delle allucinazioni. Tale comportamento, sopratutto in ambito biomedico, non è da sottovalutare: nell'esempio precedente, tra causa e cura, è presente una sostanziale differenza semantica. D'altra parte la *chatbot* non potrebbe rispondere al medico con una struttura a grafo, mentre sarebbe opportuno farlo con testo in linguaggio naturale;
- la produzione di frasi artificiali condizionate a partire da conoscenza relazionale complessa, detta anche *controllable generation* [13];
- l'uso come modulo in reti neurali più complesse per task *NLP* quali *single/multi-document summarization*.

Il lavoro presentato in questa tesi fornisce i seguenti contributi:

1. costruzione del primo dataset altamente comprensivo per il task di event graph verbalization in formato *text-to-text*, con codifica *formale* di grafi basata su testo aumentato;
2. tracciamento di una baseline sul nuovo dataset, incentrata su *LM* attualmente *SOTA* nella generazione di testo;
3. implementazione Flax [14] di un'architettura T5 personalizzata con introduzione di *linear attention*, tratta dal recente paper *Performer* [15];
4. dimostrazione dell'utilità degli eventi per il miglioramento della capacità di comprensione dei *LM* correnti in modo indipendente dal task

NLP considerato, con focus su Multi-task learning e single-document summarization.

Il documento è così organizzato:

1. nel Capitolo 1 vengono presentate le nozioni teoriche utilizzate nella tesi;
2. nel Capitolo 2 si descrivono le soluzioni *SOTA* esistenti in riferimento ai task *event extraction* e *data-to-text*;
3. nel Capitolo 3 si descrive il processo di costruzione per il dataset di *event graph verbalization* e se ne illustra la struttura. Successivamente si discute l'applicazione di T5 e BART sul dataset ottenuto, presentando i risultati di esperimenti e conferonti prestazionali, definendo una baseline;
4. infine in *Conclusioni e sviluppi futuri* si traggono le considerazioni finali e si riportano possibili direzioni che è possibile intraprendere.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Framework teorico

Questo capitolo presenterà ai lettori le nozioni teoriche che verranno usate nei capitoli successivi.

1.1 Language model basati su transformer

L'architettura transformer è stata introdotta da Google nel 2017 con il paper *Attention is all you need* [17]. Essa rappresenta una soluzione *SOTA* per quanto riguarda modelli che trasformano una generica sequenza (in questo caso un testo) in un'altra sequenza[1]. Oggigiorno i transformer sono usati per risolvere diversi task, ad esempio in NLP o Computer Vision. Sono composti da due componenti: un encoder e un decoder. Il primo si occupa di rappresentare la sequenza in input in uno spazio ad elevata dimensionalità, creandone un embedding in un vettore n-dimensionale. Il secondo serve per decodificare un embedding in una rappresentazione nello spazio originale. Nello specifico l'encoder prende in input un elemento alla volta. La struttura del transformer è illustrata in Figura 1.3.

1.1.1 Attention

In quasi tutti i task *NLP* è importante cogliere le dipendenze tra le varie parti del testo: difatti se prendiamo come esempio la frase *"Oggi ho guidato in pista un'auto da corsa. Era davvero performante!"* In questo caso il cervello umano è chiaramente in grado di cogliere la relazione semantica tra *auto da corsa* e la frase successiva. Nelle reti neurali che processano sequenze di dati, come ad esempio una *Recurrent Neural Network* o una *Long Short Term Memory* [18] (*LSTM*) vi è un problema importante in tal senso: il numero

[1] Questa tipologia di modello è anche chiamata *seq2seq*.

elevato di iterazioni non consente di cogliere in maniere efficace le relazione tra parole, difatti iterazione dopo iterazione tali reti perdono memoria delle parole viste in precedenza, rendendo superflui i loro utilizzi in presenza di testi sufficientemente lunghi. Questo problema è stato efficientemente risolto attraverso il meccanismo dell'*attention*, che proprio per il motivo citato poc'anzi rappresenta probabilmente l'elemento più importante nell'architettura transformer. Esso consente – per ogni elemento della sequenza – nell'identificare gli elementi semanticamente più affini. Permette perciò di simulare il processo che quotidianamente effettuiamo inconsciamente. L'attention consiste nel calcolare inizialmente 3 vettori per ogni parola partendo dal suo embedding: vengono prodotti i vettori Query, Key e Value. Prendiamo per esempio una semplice frase di 2 parole: *Ciao Lorenzo*. Ora si calcola un punteggio rispetto a tutte le altre parole: questo valore rappresenterà quanto la parola valutata sarà relazionata a quella in oggetto. Il punteggio in questione è calcolato eseguendo un prodotto tra il vettore query della parole in esame e il vettore key della parola per cui vogliamo calcolare il punteggio, come illustrato in Figura 1.1. Ad esempio, se vogliamo calcolare il punteggio di attention della prima parola *Ciao* con se stessa, eseguiremo un prodotto scalare $q_1 \cdot k_1$. Poi si esegue un'operazione di divisione con 8 (essendo 8 la radice quadrata di 64, cioè la dimensione di un vettore key). Si è notato che questa operazione rende stabile la discesa del gradiente. Per rendere tali valori interpretabili come probabilità in un intervallo di valori $[0, 1]$, si usano come input per una funzione *softmax*. Successivamente si moltiplica ogni vettore value per il punteggio normalizzato in $[0, 1]$. In questo modo si preserva il valore di ogni parola, diminuendo contemporaneamente l'importanza di quelle irrilevanti. Infine si sommano tutte le parole pesate tra loro, producendo il valore di attention per la prima parola. In Figura 1.1 l'illustrazione del calcolo dell'attention in forma vettoriale oppure in forma matriciale in Figura 1.2.

1.1.2 Encoder e decoder

Prendendo come esempio uno dei task più importanti, *machine translation*, l'encoder riceve come input il testo da tradurre nella lingua di partenza. Per ogni token viene calcolato un valore di *attention* che misuri quanto ogni altro token sia significativo rispetto all'attuale. Finito il processo di codifica, viene iniziato il processo di decodifica tramite il decoder. Esso riceve un prefisso iniziale, che comunica al decoder di iniziare a eseguire la traduzione. Difatti l'obiettivo è di predire il token successivo usando come input l'output dell'encoder. Dato che l'output deve considerare solo i token già generati, il decoder applica una maschera di oscuramento ai token successivi, tecnica chiamata *masked self-attention*. È importante precisare che – come si evidenzia in Figura 1.3 –

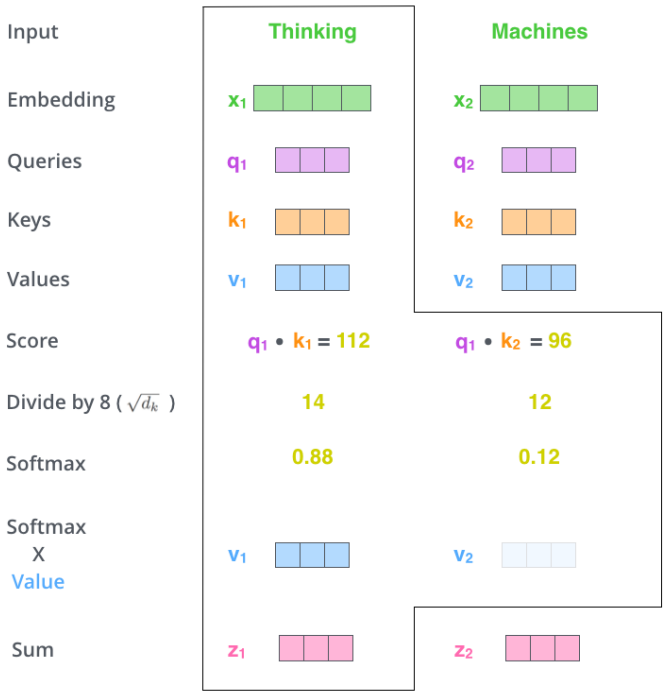


Figura 1.1: Attention in *Tranformers* in forma vettoriale.
Fonte: `http://jalammar.github.io/images/t/self-attention-output.png`

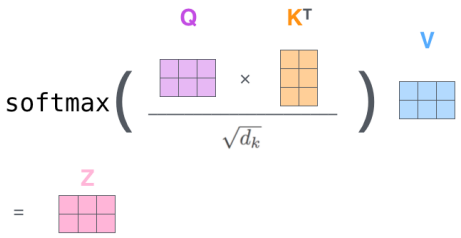


Figura 1.2: Attention in *Tranformers* in forma matriciale.
Fonte: `http://jalammar.github.io/images/t/self-attention-matrix-calculation-2.png`

sono presenti più *head*, che consentono di parallelizzare il calcolo dei valori di attention di un token per ogni altro token, dato che questi sono indipendente tra loro.

Come detto in precedenza, l'architettura transformer è alla base di modelli molto complessi, come GPT di OpenAI – di cui esistono le versioni 1 [19], 2 [20] e 3 [3] (quest'ultima è closed-source) – T5 [1] e BERT [21] di Google.

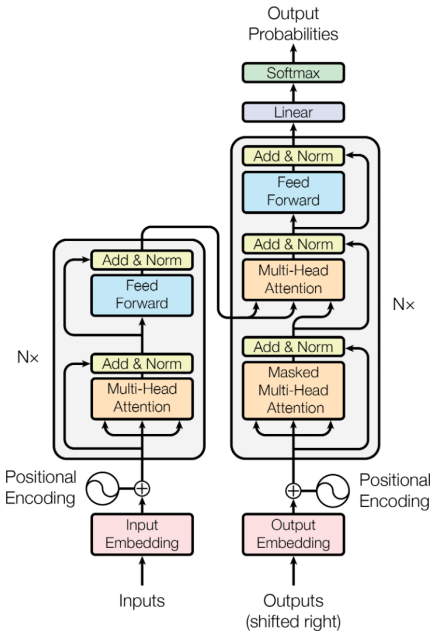


Figura 1.3: Architettura *tranformer*.
Fonte: [17]

1.2 Event Extraction

Uno dei task più importanti in ambito *NLP* è l'estrazione di connessioni semantiche (anche dette relazioni) tra entità nel testo. In ricerca, le soluzioni tradizionali di *relation extraction* (*RE*) si limitano, nella quasi totalità dei casi, a predire legami binari e piatti, cioè coinvolgenti solo due entità senza ulteriori nidificazioni all'interno dei partecipanti, i.e. in "se sovra espresso, il gene X causa la tumorigenesi" verrebbe individuata la relazione tra "gene X" e "tumorigenesi", tralasciando aspetti semantici necessari alla corretta interpretazione del contenuto, in questo caso "se sovra espresso". Ciò ha portato alla necessità di estrarre relazioni n-arie, anche definite eventi. L'obiettivo del task di *event extraction* è di classificare il tipo di ogni evento e il ruolo semantico con cui ogni partecipante vi contribuisce. Si può ridefinire il task come l'estrazione automatica di interazioni complesse (strutturate e non ambigue) da testo grezzo (per natura, non strutturato e ambiguo), i.e., *semantic parsing* [22]. Un evento è formato da un trigger – cioè l'elemento del testo che lo innesca – e un insieme di argomenti. Ogni argomento può essere un'entità o – nel caso di estrazione innestata – un altro evento. Gli eventi possono essere classificati come a dominio chiuso o aperto. Nel primo caso vi è uno schema rigido, che specifica le possibili tipologie. Nel secondo – quando non vi sono vincoli – si parla di dominio aperto, eventi ricercati, partecipanti attesi e cardinalità ammesse. Nel corso degli anni

sono state proposte diverse soluzioni in letteratura. In Figura 1.4 è presentato un esempio di predizione sul task di *event-extraction*.

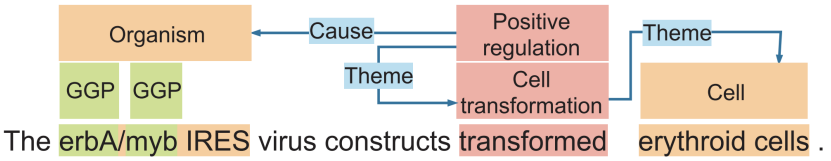


Figura 1.4: Esempio di predizione di un evento sul task di *event-extraction* in ambito biomedico.

Fonte: [23]

Tipicamente queste soluzioni risolvono il task in due passaggi: prima di tutto viene identificato il *trigger* e classificano il tipo di evento. Solo successivamente si procede all'identificazione degli argomenti e classificazione del ruolo semantico con cui partecipano all'evento. In tale *pipeline* i sotto-task vengono affrontati con classificatori indipendenti posizionati uno di seguito all'altro, interrompendo la backpropagation, ignorando interdipendenze tra sotto-task, e amplificando l'errore per ogni classificatore. Le architetture *joint* aggrediscono il problema in modo *end-to-end*, risolvendo i punti deboli del primo approccio.

Alcune soluzioni – in ambito biomedico – si basano su metodi *supervised* sia di *machine learning* che di *deep learning*. Nella prima categoria si trovano soluzioni che sfruttano *Support Vector Machines* (*SVM*) [24, 25] e *structured predictions* [26]. Nella seconda soluzioni che sfruttano *GNN* [27], *CNN* e *RNN* (descritte successivamente). Approcci ibridi sono proposti in [28]. Un altro metodo *supervised* è costituito da *manual pattern construction* [29]. Alcuni metodi *semi-supervised* includono *pattern learning* [30], *self-training* [31], *distant supervision* (che consente di definire label per dati non etichettati attraverso euristiche) [32], *LM* pre-addestrati come [23] e approcci basati su *QA* [33]. Un approccio *unsupervised* è usato in [34]. Le soluzioni proposte di *Convolutional Neural Networks* (*CNN*) [35, 36] dividono il testo in input in parole, di cui vengono generati degli *embedding*. Sulla sequenza di embedding così ottenuta viene applicata la convoluzione, considerando piccoli gruppi di embedding consecutivi, ottenendo un nuovo valore per ognuno. Oltre a questo, la natura propria della convoluzione, cioè il limitarsi a considerare intorni ristretti di *embedding*, porta a non sfruttare le interazioni che vi sono tra elementi lontani nel testo, peggiorando le performance. Di conseguenza nuove soluzioni sfruttano *RNN* [37, 38, 39], in particolare con componenti *LSTM*. In questo caso la sequenza di output contiene (per ogni termine) un valore che indica se si tratta di un *trigger* o di un argomento ed eventualmente di che tipo. Risolvendo il task in un unico passaggio si riducono le problematiche citate in precedenza, come ad

esempio lo sfruttare a pieno il contesto comune a *trigger* ed eventuali *argomenti*. Le *RNN*, rispetto alle *CNN*, riescono a considerare intorni più ampi di parole e – nonostante non elimini completamente la problematica – producono di certo migliori performance. Tali limiti vengono superati da soluzioni basate su *transformer* che – come per molti altri task in ambito *NLP* – hanno ottenuto i migliori risultati. Ad esempio, in *SciBERT* [40] vengono usati tre layer neurali per effettuare l'estrazione di entità, *trigger* e argomenti in un unico passaggio.

1.3 T5

T5 [1] (*The Text-to-Text Transfer Transformer*) è un nuovo *LM*, sviluppato da Google AI, basato su architettura transformer, in grado di gestire qualsiasi tipo di task *text-to-text*. Infatti, la sua creazione deriva dal bisogno di avere a disposizione un'unica architettura in grado di apprendere molteplici task *NLP*: questo è possibile grazie al casting di ogni task a un problema di conversione da testo a testo. Il team Google AI ha notato come modelli seq2seq di tipo *text-to-text* rappresentino la scelta migliore quando si necessita di un'architettura in grado di gestire diverse tipologie di task, ad esempio di classificazione o regressione, usando una funzione di *loss* che minimizza la lunghezza della sequenza di testo in output, rendendo al contempo più efficiente la fase di training. Quando T5 è stato definito, gli autori si sono domandati quali performance potessero essere raggiunte da una singola architettura indipendente dal task – rispetto ad architetture specifiche (ad esempio – incentrato sull'inserimento in coda di layer ad-hoc per il task di interesse). Gli esperimenti hanno fornito risultati sorprendenti, consentendo a un modello T5 di affrontare anche problemi di regressione, senza registrare cali significativi di performance. Inoltre, l'eventuale mancanza di *T5* porterebbe tipicamente ad aggiungere *layer task-specific* in coda a *LM* – come ad esempio *BERT* – rendendo più difficoltoso l'adattamento di un modello a nuovi task.

La discussione generale su T5 verte su questi argomenti:

- architettura;
- pre-training;
- fine-tuning.

1.3.1 Architettura

L'architettura di T5 implementa una classica architettura transformer[2] con qualche eccezione:

[2]Sezione 1.1

- il *LayerNorm* è stato semplificato;
- il *dropout* è aggiunto al segmento FeedForward, alle *residual connection*, al meccanismo di attenzione e agli I/O di tutta l'architettura;
- è stato applicato un *relative position encoding* invece che un *sine-based encoding*. Di conseguenza viene prodotto un encoding basato esclusivamente sull'offset tra due token. In altre parole, l'embedding si differenzia negli stessi token indipendentemente dalla loro posizione nella sequenza in input.

1.3.2 Pre-training

T5, nella sua versione originale, è stato pre-addestrato usando tecniche di parallelismo (sia a livello di modello che di dati), usando hardware all'avanguardia come multi-rack supercomputer o Google Cloud TPUs. In particolare sono stati usate 1024 TPU. Nel dettaglio, il modello è stato pre addestrato usando 2^{19} steps con un learning rate dinamico

$$\eta = \frac{1}{\sqrt{max(n,k)}}$$

con n il numero dell'iterazione attuale e k il numero di steps di warmup. Ad esempio con $k = 10^4$, $\eta = 0.01$ per i primi 100 steps, poi decresce esponenzialmente. Il dataset usato nel pre-training è stato costruito per via della mancanza di altri dataset affidabili, filtrati o ritenuti adeguati. Ne è stato ottenuto uno a partire *Common Crawl*, generato a partire da testo presente su internet e dalla dimensione via via crescente di 20TB ogni mese. Da esso è stato eliminato tutto il contenuto non testuale. Inoltre, per filtrare ulteriormente il dataset, sono state usate le seguenti euristiche:

- vengono incluse solo le frasi che finiscono con un segno di punteggiatura terminale (sono .,!?");
- vengono escluse le pagine che contengono meno di 5 frasi, che a loro volta devono essere composte di almeno 3 parole;
- vengono escluse le pagine che contengono parole presenti in *List of Dirty, Naughty, Obscene or Otherwise Bad Words*;
- vengono escluse le pagine che contengono la parola *Javascript*: potrebbe essere codice non identificato durante la creazione di *Common Crawl dataset*;

- vengono escluse le pagine che contengono *lorem ipsum*: potrebbe trattarsi di pagine di default, percui non significative;
- vengono escluse pagine contenenti parentesi graffe: potrebbe trattarsi di codice;
- vengono rimossi i duplicati, eseguendo la ricerca su tutti gli span di testo composti da tre frasi;
- vengono scartate le pagine che al 99% di probabilità non sono in lingua inglese.

Il dataset ripulito prende il nome di *Colossal Clean Crawled Corpus* o *C4*[3], ha una dimensione di circa 750GB di linguaggio naturale ripulito ed è rilasciato su *TensorFlow Datasets*[4]. C4 è stato considerato migliore rispetto a un dataset più filtrato (e quindi di dimensioni più contenute), tranne nei casi in cui venga effettuato fine-tuning su task di domini molto specifici. Altre evidenze suggeriscono però come l'aggiunta di ulteriori filtri al dataset originale e porti il modello a un calo drastico di performance: è quindi necessario – per motivazioni principalmente legate ai costi – un bilanciamento tra qualità del modello e dimensione del dataset usato. In quest'ottica, alcuni esperimenti hanno evidenziato come un modello più complesso (di dimensione maggiore) – addestrato con poche epoche su un dataset esiguo – sia migliore di un modello più semplice (di dimensione minore) addestrato con molte epoche su un dataset più corposo. In generale – se non dovessero sussistere problematiche di costi – l'aumento di complessità del modello e delle dimensioni del dataset possono contribuire all'aumento di performance. Esiste una terza opzione – l'*ensembling* – cioè il combinare più modelli tra loro. In questo modo vengono garantiti aumenti di performance solo nel caso in cui i modelli combinati siano stati pre-addestrati diversamente.

1.3.3 Fine-Tuning

Partendo da un modello T5 pre-addestrato, lo si può specializzare su un task specifico (downstream task), facendogli dimenticare alcuni vecchi pesi e facendogliene imparare degli altri: questo processo prende il nome di fine-tuning o più semplicemente training[5]. Il fine-tuning è eseguito attraverso *teacher forcing* e l'ottimizzazione della funzione obiettivo *maximum likelihood*, indipendentemente dal task. T5 è stato originariamente addestrato e valutato su diversi *benchmark dataset*, tra cui:

[3]Una documentazione per il corpus C4 è stata recentemente proposta da Dodge et al. [41]

[4]`https://www.tensorflow.org/datasets/catalog/c4`

[5]Si distingue il training dal pre-training

- GLUE [42] e SuperGLUE [43];
- CNN/DM [44] per la sintesi di testo;
- SQuAD [45] per *question answering*;
- WMT [46] per *machine translation*, su coppie di lingue EN-FR, EN-GE, EN-RO.

In particolare, per eseguire fine-tuning su GLUE e SuperGLUE, T5 è stato addestrato sui seguenti task:

- Sentence acceptability judgement (CoLA [47]);
- Sentiment analysis (SST-2 [48]);
- Paraphrasing/sentence similarity (MRPC [49], STS-B, QQP);
- Natural language inference (MNLI [50], QNLI [42], RTE, CB);
- Coreference resolution (WNLI, WSC [51]);
- Sentence completion (COPA [52]);
- Word sense disambiguation (WIC);
- Question answering (MultiRC [53], ReCoRD [54]; BoolQ [55]).

Nel dettaglio, è stato eseguito fine-tuning sul modello base pre-addestrato su *C4* usando 2^{18} step con un learning rate costante di 0.001. I checkpoint sono stati salvati ogni 5000 step. L'abilità di addestrare un modello T5 su più task è chiamata *Multi-task learning* (*MTL*). In sostanza si trattano i molteplici task come un singolo task concatenando i relativi dataset tra loro. L'architettura deve però riconoscere il task da eseguire, ad esempio in fase di inferenza: all'input viene anteposto un prefisso specifico per il task. In Figura 1.5 sono disponibili alcuni esempi di input:

1.4 Pre-processing di testo

I modelli in ambito *NLP* richiedono come input un testo, che va ripulito e modificato al fine di aumentare le performance. Alcune delle tecniche usate sono:

- tokenization;
- stop words removal;

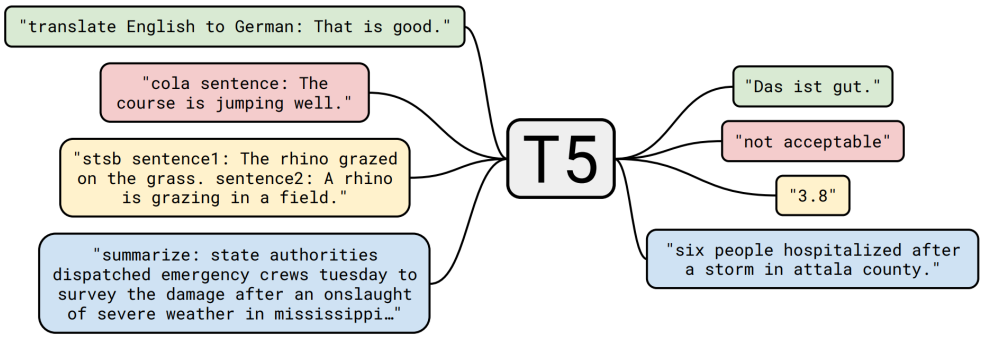


Figura 1.5: Esempi di predizioni generate da un modello *T5* addestrato in modalità *MTL*.

Fonte: [1]

- stemming;
- normalization;
- lemmatization;
- parts of speech tagging.

In particolare, al fine di questo lavoro, ci si concentrerà sulla tokenization.

1.4.1 Tokenization

Il processo di *tokenization* consiste nel dividere un testo in linguaggio naturale in piccoli insiemi, chiamati *token*. Tale processo può essere eseguito a livello di frase o a livello di parola e permette di agevolare i modelli di apprendimento successivi nella pipeline. Per esempio, la frase "It is raining. We have to stay home." può essere trasformata in un vettore di token nei modi seguenti:

- ["It", "is", "raining", ".", "We", "have", "to", "stay"', "home", "."] nel caso di una tokenizzazione a livello di singola parola;
- ["It is", "raining", ".", "We", "have", "to", "stay", "home", "."] nel caso di una tokenizzazione a livello di singola parola, usando un dizionario che riconosca ad esempio la vicinanza di 'it' e 'is';
- ["It is raining.", "We have to stay home."] nel caso di una tokenizzazione a livello di frase;

La tokenizzazione però presenta diverse sfide, tra cui individuare correttamente l'inizio e la fine delle parole. In inglese generalmente una parola è divisa da un'altra usando segni di punteggiatura o uno spazio, ma non è una caratteristica comune a tutti i linguaggi. Ad esempio in Cinese, Giapponese o Coreano le parole sono rappresentate da simboli e questo rende difficile il compito di divisione tra di esse. Anche in inglese si presentano tali problemi, ad esempio nell'utilizzo di simboli seguiti da numeri, come nel caso di unità di misure scientifiche o valute economiche. Inoltre, in diverse forme di linguaggio sono presenti delle abbreviazioni (come "I'm" al posto di "I am" in inglese). Nel tempo sono stati rilasciati strumenti sempre più evoluti in grado di eseguire efficacemente questo task, ma la ricerca tutt'ora continua.

Qui viene introdotto il concetto di n-gram, che rappresenta un sottoinsieme di parole o token consecutivi avente cardinalità n, con $n \in N$; ad esempio un 1-gram rappresenta i token/parole singoli, un *2-gram* raggruppamenti di 2 token/parole consecutivi e così via.

1.5 Metrica di valutazione ROUGE

Valutare le performance di un modello in ambito NLP, ad esempio di generazione del testo, usando le metriche di un problema di classificazione può essere problematico, in quanto potrebbero condurre a risultati poco veritieri. In questo scenario poco rassicurante è stata introdotta *ROUGE* [56], acronimo di Recall-Oriented Understudy for Gisting Evaluation. ROUGE può essere di diversi tipi, di seguito verranno esplorate:

- ROUGE-N;
- ROUGE-L;
- ROUGE-S.

1.5.1 ROUGE-N

In questa versione di ROUGE viene misurato il numero di *n-gram* in comune tra il testo prodotto in output e quello reale. In altre parole si valuta quanto il testo prodotto in output dal modello rispecchi il miglior testo prodotto da un essere umano, se ad esempio si sta eseguendo un task di machine translation. Nella pratica spesso sono usate ROUGE-1, ROUGE-2 e ROUGE-3, che usano rispettivamente 1-gram, 2-gram e 3-gram. La divisione ulteriore sta nella tipologia di calcolo, infatti si posso usare recall, precision o F1-score, ognuno dei quali presenta un diverso significato. Recall misura il numero di *n-gram* in

comune tra il testo generato in output dal modello e quello reale, e lo divide per il numero di *n-gram* nel testo reale:

$$Recall = \frac{conta(ngrams_{predicted} \cap ngrams_{real})}{conta(ngrams_{real})}$$

Ad esempio, la Figura 1.6[6] evidenzia come la recall raggiunga il 100% grazie alla capacità del modello di catturare tutti gli *n-gram* all'interno del testo reale. Questo capita anche nonostante vi siano ulteriori *n-gram* non presenti nel testo reale.

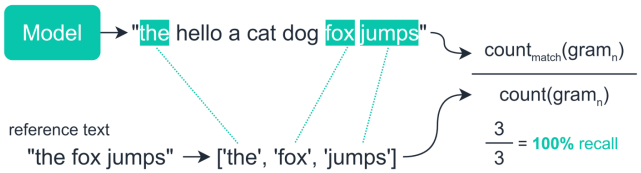


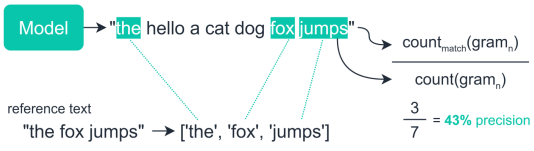
Figura 1.6: ROUGE-N calcolata su recall.

Precision misura il numero di *n-grams* in comune tra il testo generato in output dal modello e quello reale, e lo divide per il numero di *n-grams* nel testo generato:

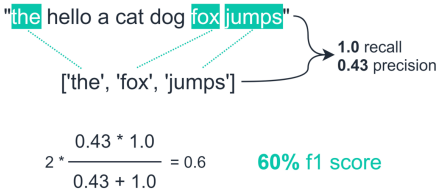
$$Precision = \frac{conta(ngrams_{predicted} \cap ngrams_{real})}{conta(ngrams_{predicted})}$$

In Figura 1.7a si evidenzia come la precision dell'esempio precedente sia il 43%, valore molto più basso rispetto a quello di recall. F1-score, come si evince da Figura 1.7b, rappresenta la capacità del modello di generare testi corretti (elevata recall) a fronte di un numero di n-gram il più possibile conforme a quello del testo originale (elevata precision). Fa ciò unendo le due metriche insieme.

$$F1 - score = 2 * \frac{recall * precision}{recall + precision}$$



(a) ROUGE-N calcolata su precision.

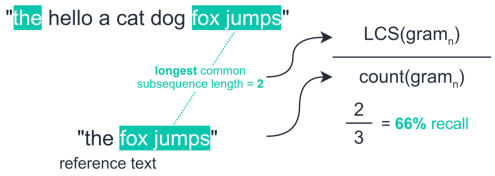


(b) ROUGE-N calcolata su F1-score.

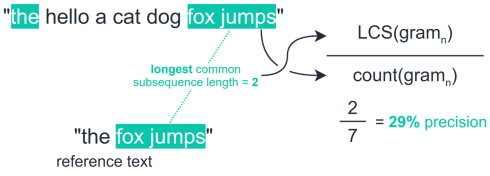
[6]fonte immagini ROUGE: https://towardsdatascience.com/the-ultimate-performance-metric-in-nlp-111df6c64460

1.5.2 ROUGE-L

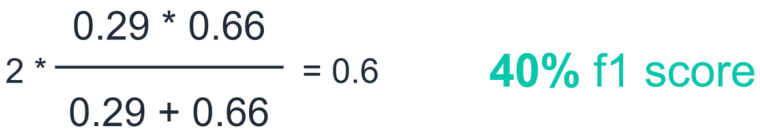
In tale versione di ROUGE, viene misurata la più lunga sequenza in comune (LCS è un acronimo di *longest common subsequence*) tra il testo generato in output e quello reale. Questo in base all'assunto che una più lunga sequenza in comune sia indice di maggior similarità tra i testi. In Figura 1.8a, Figura 1.8b e Figura 1.8c vengono evidenziati il calcolo di recall, precision e F1-score rispettivamente, sostituendo il conteggio degli *n-grams* in comune con una funzione *LCS*.



(a) ROUGE-L calcolata su recall.



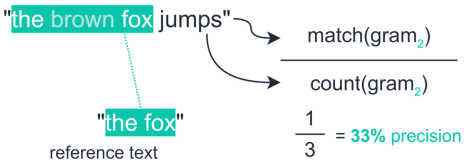
(b) ROUGE-L calcolata su precision.



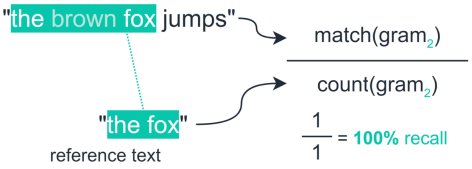
(c) ROUGE-L calcolata su F1-score.

1.5.3 ROUGE-S

L'ultima tipologia di ROUGE trattata, meno usata nella pratica rispetto alle altre due, permette di introdurre un certo grado di *latenza* rispetto a ROUGE-N, includendo la possibilità di considerare *n-grams* sovrapposti ma non consecutivi: da ciò deriva il nome ROUGE - S (skip-gram). In Figura 1.9a, Figura 1.9b vengono presentati esempi di calcolo di recall e precision.



(a) ROUGE-S calcolata su recall.



(b) ROUGE-S calcolata su precision.

Capitolo 2

Soluzioni esistenti

2.1 DeepEventMine

DeepEventMine [23] rappresenta una soluzione *SOTA* per quanto concerne il task di event extraction in ambito biomedico (fondamentalmente inverso a quello principalmente trattato in questa tesi).

2.1.1 Introduzione

Un evento consiste di un trigger e zero o più argomenti. Un trigger è un elemento del testo, tipicamente un verbo (i.e. "regulates") o un verbo normalizzato (i.e. "regulation"), che denota la presenza di un evento. Ogni argomento è un'entità o un altro evento ed ha un ruolo che caratterizza il suo collegamento all'evento. Ogni evento è chiamato *nested* se ha altri eventi all'interno del set degli argomenti, mentre è chiamato *flat* se vi sono solo entità nel set degli argomenti. L'architettura è presentata in Figura 2.1.

2.1.2 BERT layer

Il BERT layer riceve una sequenza di sotto-parole e assegna una rappresentazione a ciascuna di esse. Assumiamo che ogni sequenza S abbia n parole e che la parola i sia divisa in s_i sotto-parole. Questo layer assegna un vettore $v_{i,j}$ alla sotto-parola j della parola i. Produce inoltre la rappresentazione dell'intera frase v_s, che corrisponde all'embedding CLS (classification) prodotto da BERT.

2.1.3 Entity trigger layer

L'entity/trigger layer assegna, per ogni possibile sequenza di parole (potenzialmente sovrapposte tra loro), il tipo di entità o trigger, se presenti. Il

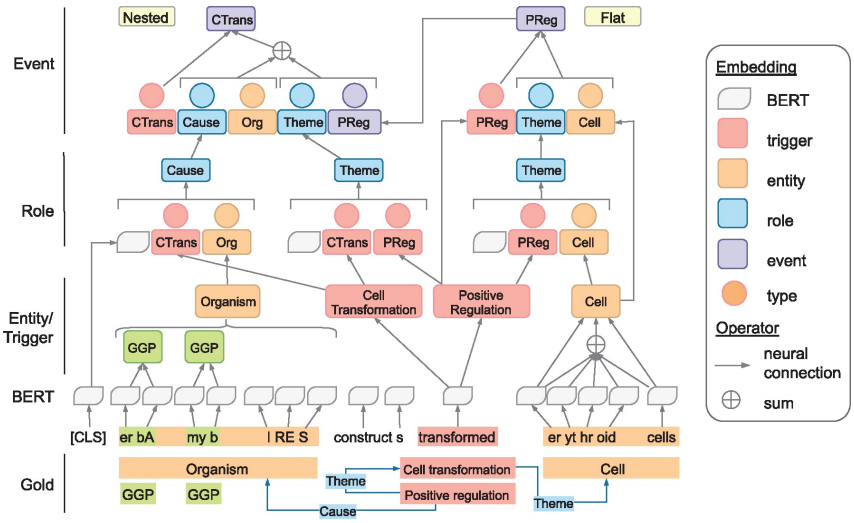


Figura 2.1: Architettura di DeepEventMine.
Fonte: [23]

modello usato più frequentemente è *NER*, introdotto in [21]. Quest'ultimo considera solo la prima sotto-parola di ogni parola. Per superare questo limite, gli autori hanno utilizzato il modello [57], che permette di sfruttare tutte le informazioni di ogni parola.
Nello specifico questo layer considera tutte i possibili span (intervalli) di parole all'interno della frase, con lunghezza minore o uguale a due iperparametri: L_{entity} per le entity e $L_{trigger}$ per i trigger.

2.1.4 Span representation learning

Lo span dalla parola k alla parola l è rappresentato dalla concatenazione di 3 vettori:

$$m_{k,l} = [v_{k,1}; \frac{\sum_{i=1}^{l}\sum_{j=1}^{s_i} v_{i,j}}{\sum_{i=k}^{l} s_i}; v_{l,s_l}]$$

1. la rappresentazione vettoriale della prima sotto-parola della parola iniziale k;

2. somma vettoriale, per ogni parola dello span, dei vettori corrispondenti ad ognuna delle sue sotto-parole. Il vettore risultate è diviso per uno scalare, vale a dire il numero totale di sotto-parole nello span;

3. la rappresentazione vettoriale dell'ultima sotto-parola della parola finale l.

Ogni span potrebbe essere associato a più tipi, quindi si procede usando più classificatori binari (ognuno associato a un tipo diverso) e si selezionano i tipi corrispondenti ai modelli i cui output sono maggiori del 50%.

2.1.5 Role layer

Dati i trigger e le entità ottenute dai classificatori binari nel livello precedente, questo layer enumera tutte le coppie trigger - argomento (possono essere trigger-entity o trigger-trigger). Ad ogni coppia deve essere associato un ruolo, se presente. Come detto in precedenza, ogni ruolo è composto da un trigger ed un argomento, quindi per prima si calcolano le rappresentazioni di tutti i trigger e argomenti individuati dal layer entity/trigger. Le rappresentazioni di questi due tipi di elementi sono calcolate nello stesso modo. Un trigger t (o similmente un argomento a), che inizia dalla parola t_{start} e finisce alla parola t_{end}, è rappresentato dalla concatenazione di due vettori:

1. rappresentazione dello span dalla parola $k = t_{start}$ alla parola $l = t_{end}$;
2. embedding s_t.

In definitiva, queste sono le rappresentazioni vettoriali di un generico trigger t e di un generico argomento a.

$$v_t = [m_{t_s,t_e}; s_t]$$

$$v_a = [m_{a_s,a_e}; s_a]$$

Nel frattempo, la rappresentazione del contesto C è ottenuta dalla rappresentazione dell'intera frase ottenuta dal layer BERT. In questo caso $C = v_s$. Nelle fasi sperimentali si sono notati aumenti di performance del modello nel momento in cui veniva usato il contesto della frase.

La rappresentazione della generica coppia trigger - argomento i è $r_i \in \mathbb{R}^{d_r}$. Si usa la rappresentazione del trigger v_t, la rappresentazione dell'argomento v_a e la rappresentazione del contesto c:

$$r_i = GELU(W_r[v_t; v_a; c] + b_r)$$

in cui sia W_r (vettore dei pesi) che b_r (bias) sono parametri da apprendere e $GELU$ è la funzione di attivazione *Gaussian Error Linear Unit* [58]. Ora si vuole predire il ruolo corrispondente alla generica coppia i: la sua rappresentazione r_i viene usata come input ad un layer interamente connesso, avente una funzione di attivazione *softmax*, per predire il ruolo corrispondente alla coppia.

2.1.6 Event layer

Dati i trigger e le entità individuate nella frase dall'entity/trigger layer e le coppie individuate dal role layer, l'event layer deve enumerare tutte le combinazioni legali di coppie di ruoli, per ottenere degli eventi candidati per ogni trigger (un evento candidato può anche avere zero argomenti). Ogni evento candidato è poi classificato come *evento* o *non evento*. In più, questo layer è in grado di individuare modificatori di speculazione o negazione.

Per identificare eventi *nested*, si usa un approccio bottom-up. Nel dettaglio, la classificazione avviene nel seguente ordine:

- Si classificano prima eventi candidati che non contengono alcun trigger all'interno del set degli argomenti (possono non avere alcun argomento o entità). In questo modo si ottengono eventi *flat*.

- Poi si classificano eventi candidati con almeno un trigger come argomento. Durante il processo si sostituiscono ai trigger i relativi eventi. Nel caso vi siano più trigger come argomenti si duplica il trigger padre tante volte quante il numero di trigger figli.

Classificando in quest'ordine si ottiene un approccio bottom-up, essendo che gli eventi *nested* già predetti sono usati nella classificazione degli eventi candidati di ordine superiore. Il processo termina quando ognuno degli eventi candidati è classificato come *evento* o *non evento*.

La rappresentazione di un generico evento candidato i è $e_i \in \mathbb{R}^{d_e}$ ed è calcolata partendo dalla rappresentazione del trigger t, del ruolo r, dell'array n_t-dimensionale relativo al tipo di ruolo u e di tutti gli argomenti $a_* \in e_a$. Nel dettaglio:

$$e_i = [v_t; \sum_{j \in e_a} GELU(W_p[r_j; u_j; v_{a_j}] + b_p) + \sum_{j \notin e_a} GELU(W_n[r_j; u_j; v_{a_j}] + b_n)]$$

Dove:

- Prima sommatoria: W_p e b_p sono parametri del modello che rappresentano rispettivamente il vettore dei pesi e il bias dei ruoli individuati dal role layer e che sono inclusi all'interno dell'evento i;

- Seconda sommatoria: W_n e b_n sono parametri del modello che rappresentano rispettivamente il vettore dei pesi e il bias dei ruoli individuati dal role layer e che condividono i trigger ma che non sono inclusi all'interno dell'evento i.

Includendo sia i ruoli presenti che quelli non presenti all'interno della struttura dell'evento i, si usano tutti i possibili set di ruoli per scegliere quello più appropriato. Quando l'evento candidato e_i ha un trigger e un evento come argomento si sostituisce v_{a_j} con la rappresentazione dell'evento candidato e_i. Prima di effettuare la sostituzione, la rappresentazione dell'evento e_i viene passata a un hidden layer con funzione di attivazione $GELU$ per produrre e'_i. e'_i è poi passata ad un classificatore binario (*evento* o *non evento*). Nel caso l'evento candidato e_i' sia stato classificato come evento può essere inviato ad un ulteriore modello di classificazione con funzione di attivazione *softmax* per allegare ad esso alcune ulteriori proprietà, quali speculazione o negazione.

2.2 Event-extraction con T5

Il framework *Translation between Augmented Natural Languages* o *TANL* [59], sviluppato da Amazon AWS, ha permesso di risolvere molti task NLP, inclusi – tra i tanti – l'*entity*, *relation* ed *event* extraction. Invece di creare un'architettura singola per ogni task è stato deciso di sfruttare il vantaggioso *MTL* offerto da T5. Infatti il problema complesso dell'*event extraction* viene convertito a un problema *text-to-text*: l'output consiste in un input aumentato con le informazioni richieste al modello. Per quanto concerne il modello di event extraction proposto dagli autori, vengono prima estratti i trigger e classificati con la propria tipologia. Successivamente, considerando un trigger alla volta, vengono individuati i relativi argomenti. Di seguito un esempio di input/output di un testo in cui sono presenti due trigger:

Sotto-task EE	**Testo target aumentato**
Trigger extraction input	Two soldiers were attacked and injured yesterday.
Trigger extraction output	Two soldiers were **[attacked | *attack*]** and **[injured | *injury*]** yesterday.
Argument extraction input	Two soldiers were [attacked | attack] and injured yesterday.
Argument extraction output	**[Two soldiers | individual | target = *attacked*]** were attacked and injured **[yesterday | time | attack time = *attacked*]**

Tabella 2.1: Esempi di predizioni generate da un modello *T5* sul task di *event-extraction*.

2.3 TEKGEN Data-To-Text

TEKGEN [11], presentato da Google, è un modello *SOTA* in grado di verbalizzare l'intero English Wikidata *KG*, ovvero il grafo ottenuto da Wikipedia. In Figura 2.2 è illustrato un esempio di verbalizzazione.

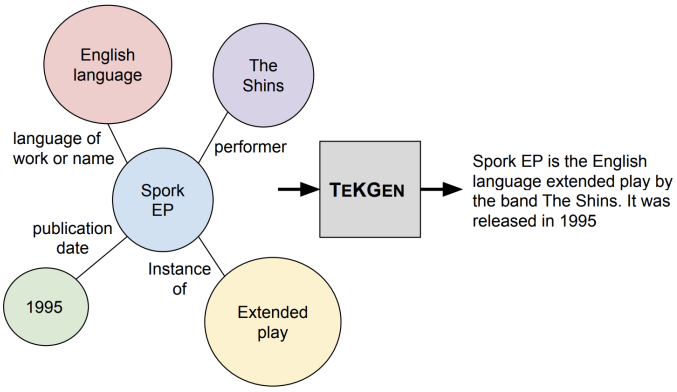


Figura 2.2: Esempio di verbalizzazione di KG con TEKGEN.
Fonte: [11]

Il task è stato risolto in più fasi:

1. le triple (soggetto, predicato, oggetto) del *KG* di *Wikipedia* vengono allineate con testo in linguaggio naturale usando *distant supervision*;
2. *T5* viene addestrato in serie prima sul *corpus* principale ottenuto al passo precedente e poi – seppur in pochi *step* – sul *corpus WebNLG* [60];
3. *BERT* viene addestrato per ottenere uno punteggio di qualità semantica per il testo generato in output.

Capitolo 3

Contributi

Nel capitolo che segue verranno presentati i contributi introdotti da questa tesi. In particolare verrà definito il dataset usato nel task di *event graph verbalization* – affrontato sia con *T5* che con *BART* – e verranno mostrati al lettore i risultati degli esperimenti condotti.

3.1 Dataset

Struttura

Un contributo è stato il dataset usato per eseguire fine-tuning su un task di tipo *Data-to-Text*, in particolare da strutture dati a grafo. Per fare ciò è necessario:

- creare un dataset che associ a un evento la relativa *event mention*, ovvero lo span di testo descrivente l'evento e tutti i suoi componenti;
- codificare formalmente un *event graph* in forma di stringa.

Per perseguire questi due obiettivi è stato necessario munirsi di dataset già usati per eseguire event-extraction e riadattarli allo scopo. Proprio per questo motivo sono stati usati i dataset impiegati nell'addestramento di DeepEventMine (Sezione 2.1). Essi sono descritti in Tabella 3.1.

Le annotazioni gold riportate all'interno di un dataset possono essere espresse mediante uno tra due possibili formalismi:

1. **.a***: il più usato, è suddiviso in:
    - .a1: entità gold, cioè entità che partecipano ad eventi date in input all'estrattore;

Dataset	Sorgenti	Tipi individuati	Elementi individuati
Genia Event Corpus [61]	1000 abstracts	35 tipi di entità, 35 tipi di eventi	93,293 proteine, 36,114 eventi
Genia Event 2011 (GE2) [62]	1,210 abstracts, 14 full papers	2 tipi di entità, 9 tipi di eventi, 2 tipi di modificatori ↪ negation, speculation	21,616 proteine, 18,047 eventi, 53.83% eventi nested
Epigenetics and Post-translational Modifications (EPI) [63]	1200 abstracts	2 tipi di entità, 14 tipi di eventi, 2 tipi di modificatori ↪ negation, speculation	15,190 proteine, 3,714 eventi, 369 modificatori, 8.27% eventi nested
Infectious Diseases (ID) [64]	30 full papers	5 tipi di entità, 10 tipi di eventi, 2 tipi di modificatori ↪ negation, speculation	12,740 entità, 4,150 eventi, 214 modificatori, 41.50% eventi nested
Multi-Level Event Extraction (MLEE) [65]	262 abstracts	16 tipi di entità, 19 tipi di eventi	8,291 entità, 6,667 eventi, 53.17% eventi nested
GENIA-MK [66]	1000 abstracts	5 tipi di modificatori ↪ knowledge type, ↪ certainty level, ↪ polarity, manner, source	36,858 eventi
Genia Event 2013 (GE3) [67]	34 full papers	2 tipi di entità, 13 tipi di eventi, 2 tipi di modificatori ↪ negation, speculation	12,068 proteine, 762 entità, 9,364 eventi, 58.89% eventi nested
Cancer Genetics (CG) [68]	600 abstracts	18 tipi di entità, 40 tipi di eventi, 2 tipi di modificatori ↪ negation, speculation	21,683 entità. 17,248 eventi, 1,326 modificatori, 50.76% eventi nested
Pathway Curation (PC) [69]	525 abstracts	4 tipi di entità, 23 tipi di eventi, 2 tipi di modificatori ↪ negation, speculation	15,901 entità, 12,125 eventi, 571 modificatori, 57.57% eventi nested
Gene Regulation Ontology (GRO) [70]	300 abstracts	174 tipi di entità, 126 tipi di eventi	11,819 entità, 5,241 eventi

Tabella 3.1: Dataset impiegati – e relative proprietà – nella creazione del dataset proposto.

- .a2: predizione di eventi e di ulteriori entità non presenti nelle entità gold.

2. **.ann**: integra in un unico file sia le entità (gold e non) che gli eventi estratti. È la soluzione adottata da tool di visualizzazione come *brat*[1],

[1]https://brat.nlplab.org/

che non necessitano di distinzione tra input e output.

Modificatori

È stato aggiunto il dataset GENIA-MK per via della sua capacità di associare ad ogni evento alcuni modificatori. Alcuni di essi sono elencati in Tabella 3.2.

Modificatore	**Descrizione**	**Valori**
Polarity o Negation	Veridicità di un evento	Positive (default) Negative
Tense	Proprietà temporale	Unspecified (default) Past Future Present
Genericity	Singola occorrenza o meno	Generic (default) Specific
Modality	Reale occorrenza o meno	Other (default, e.g. Believed) Asserted
Speculation	Evento speculato o meno	False (default) True
Source	Origine della conoscenza espressa	Current paper (default) Other
Manner	Livello di intensità di un evento	Neutral (default) High Low
Certainty level	Confidenza rispetto alla veridicità di un evento	L3 (default, high) L2 (not complete) L1 (low)
Knowledge type	Tipo di informazione espressa dall'evento	Other (default) Investigation Observation Analysis Fact Method

Tabella 3.2: Esempi di modificatori.

Tale caratteristica permette di addestrare T5 non solo a produrre output in linea con l'evento, ma anche con le proprietà specificate, che ad esempio possono essere di negazione o speculazione.

Grammatica degli eventi

Vi è stata necessità di creare un formalismo in grado di convertire un grafo evento in una stringa. L'ispirazione è nata dal paper *Structured prediction as translation between augmented natural languages* [59] di Amazon AWS, che adotta le parentesi quadre per introdurre un testo aumentato, come rappresentato in Figura 3.1.

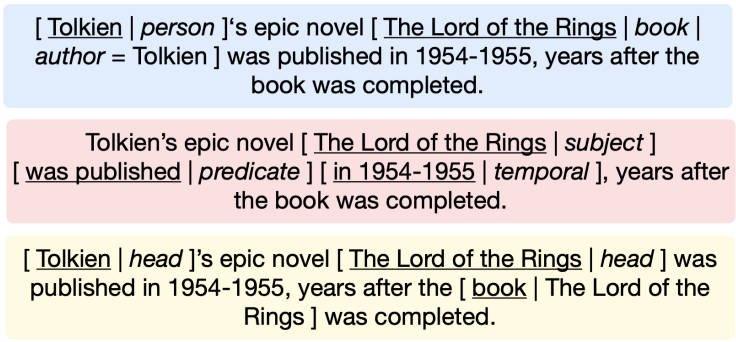


Figura 3.1: Testo aumentato con parentesi quadre.
Fonte: [59]

Per descrivere il formalismo viene definita una grammatica G. È stato usato l'applicativo *JFLAP* [71], il quale permette – tra le varie funzionalità – di testare una grammatica creata e di verificare se una certa stringa evento è sintatticamente ben strutturata e accettata dal sistema. La grammatica di un evento nel nostro caso di studio è descritta da:

$$G = \langle N, \Sigma, P, \{EV\}, S \rangle$$

L'insieme dei simboli non terminali N è riportato in Tabella 3.3. Il simbolo di partenza non terminale è EV.

Simbolo	**Descrizione**
EV	evento
T	trigger
P	partecipante
MTT	matched text trigger
MTE	matched text entity
EVT	tipo di evento
M_ext	modificatore esterno
M	modificatore
MV	valore del modificatore
E	entità
ET	tipo di entità
R_ext	ruolo esterno
R	ruolo

Tabella 3.3: Simboli non terminali N.

L'insieme delle regole di produzione (o derivazione) P è riportato in Tabella 3.4.

$EV \rightarrow TP$
$T \rightarrow [MTT \vert EVTM\_ext]$
$M\_ext \rightarrow \vert M\_ext$
$M\_ext \rightarrow M\_ext \vert M\_ext$
$M\_ext \rightarrow M = MV$
$P \rightarrow PP$
$P \rightarrow E$
$P \rightarrow EV$
$E \rightarrow [MTE \vert ET \vert R\_ext]$
$R\_ext \rightarrow R\_ext \vert R\_ext$
$R\_ext \rightarrow R = MTE$
$MTT \rightarrow matched\_text\_trigger$
$MTE \rightarrow matched\_text\_entity$
$EVT \rightarrow event\_type$
$M \rightarrow modificatore$
$MV \rightarrow valore\_modificatore$
$ET \rightarrow tipo\_di\_entita$
$R \rightarrow ruolo$

Tabella 3.4: Insieme delle regole di produzione non terminali P.

L'insieme dei simboli terminali Σ è riportato in Tabella 3.5.

matched_text_trigger
matched_text_entity
tipo_di_evento
modificatore
valore_modificatore
tipo_di_entità
ruolo

Tabella 3.5: Simboli terminali Σ.

Creazione del dataset

Sono stati eliminati sia gli eventi duplicati – presenti per via di alcune sovrapposizioni tra i dataset – sia quelli innestati aventi cicli (15 di questa tipologia, tutti provenienti dal dataset *GENIA-MK*).

```
df = pd.DataFrame(dataset).drop_duplicates(subset='event')
```

Ad ogni entry del dataset è stata aggiunta una colonna *class_for_splitting*, che associa all'evento in questione e alle sue *events mention* alcune caratteristiche. In particolare vengono associati il nome del dataset da cui è stato preso l'evento, il tipo di quest'ultimo e il numero di event mention a cui esso fa riferimento. Così facendo, si possono identificare le entry facenti parte di classi scarsamente popolate, in particolare si è scelta come soglia minima 2.

```
min_class_occurence = 2
df = df[df.groupby('class_for_splitting').class_for_splitting
    .transform('count') > min_class_occurence]
```

Il dataframe finale è usato per dividere features e output:

```
y = df['event_mention']
X = df.drop('event_mention', axis=1)
```

Splitting

La colonna *class_for_splitting* è stata necessaria per operare uno splitting stratificato, che permette di dividere in modo più intelligente training e test set. Non operando in questo modo, si rischia di testare un modello su dati molto simili a dati su cui è stato addestrato, anche se effettivamente non sono uguali, e.g. due eventi sintatticamente diversi – che però provengono dallo stesso dataset – hanno lo stesso tipo e usano lo stesso numero di event mentions risultano molto probabilmente simili tra di loro. La proporzione usata è stata 90% training set, 10% test set, diviso a sua volta in due parti uguali per validation e test set.

```
random_seed = 42
test_size_ratio = 0.1
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=test_size_ratio, random_state=random_seed,
    stratify=X['class_for_splitting'])
X_validation, X_test, y_validation, y_test = train_test_split(X_test,
    y_test, test_size=0.5, random_state=random_seed)
```

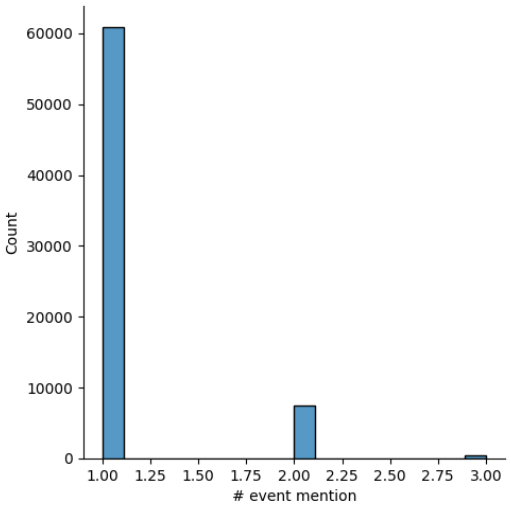
Statistiche

Le dimensioni del dataset così creato e diviso in training/validation/test sets, sono riportate in Tabella 3.6.

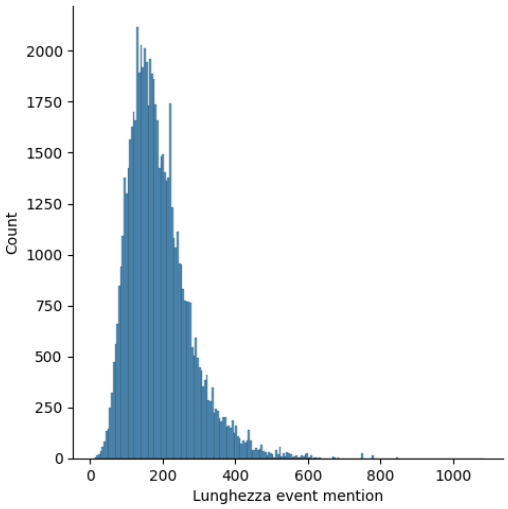
Dataset	Dimensione
Training set	61869
Validation set	3437
Training set	3438

Tabella 3.6: Dimensioni dataset.

In Figura 3.2a e Figura 3.2b sono riportate alcune statistiche sul dataset.



(a) Distribuzione del numero di *event mention* associate ad un evento.



(b) Distribuzione della lunghezza delle *event mention* associate ad un evento.

3.2 Event graph verbalization baseline

3.2.1 T5

Per eseguire fine-tuning bisogna disabilitare la pre-allocazione di memoria da parte di TensorFlow:

```
config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth=True
sess = tf.compat.v1.Session(config=config)
```

Rimuovere un task eventualmente presente dal registro di T5:

```
TaskRegistry = dataset_providers.TaskRegistry
TaskRegistry.remove("event_graph_verbalization_task")
```

Aggiungere il nuovo task al registro di T5:

```
TaskRegistry.add(
  "event_graph_verbalization_task",
  dataset_providers.TextLineTask,
  split_to_filepattern = {
      "train": os.path.join(data_dir, train_file),
      "validation": os.path.join(data_dir, validation_file)
  },
  skip_header_lines = 1,
  text_preprocessor = preprocessors.preprocess_tsv,
  metric_fns=[functools.partial(
      rouge_top_beam, beam_size=fine_tuning_cfg.beam_size)]
)
```

In particolare, per ogni task si specificano:

1. il nome;
2. i percorsi dei dataset di training e validation;
3. se eventualmente ignorare la prima riga di intestazione;
4. in base a quale formato pre-processare il testo (in questo caso TSV);
5. su quali metriche il modello è valutato (usando il validation set): in questo caso sono usate rouge1, rouge2 e rougeL (Sezione 1.5);

Infine viene lanciato il fine-tuning:

```
start = time.time()
train(model_dir="../data/model_data/linear_attention/model_checkpoints/",
↪  config=fine_tuning_cfg)
end = time.time()
elapsedTime = float((end - start) / 3600)
```

Al termine del fine-tuning viene inviata una mail di conferma:

```
email = 'nome.cognome@email.it'
message = f'echo "Training T5 for {fine_tuning_cfg.num_epochs} epochs has
↪  finished in {elapsedTime} hours" | ssmtp '
os.system(message + email)
```

Per poter inviare email è stato usato il pacchetto SSMTP, come descritto nell'Appendice A.

Esperimenti

Per eseguire gli esperimenti è stato usato un ambiente virtuale basato su container Docker (Appendice A). Per ogni esperimento effettuato è stato generato un file di configurazione, contenente gli iperparametri necessari. Le possibili varianti di T5 tra cui scegliere sono state:

- T5-small (60M parametri);
- T5-base (220M parametri);
- T5-large (770M parametri);
- T5-3B (3 miliardi di parametri);
- T5-11B (11 miliardi di parametri).

Mentre *T5-small* non garantiva performance accettabili, la versione *large* presentava una dimensione troppo elevata. Prendendo in considerazione le risorse a disposizione, la scelta naturale è ricaduta su *T5-base*, che rappresentava un giusto compromesso tra tempi di addestramento, memoria richiesta e performance.

T5-C4 con attention lineare Gli esperimenti sono stati eseguiti con una GPU GeForce RTX 3090 (24GB). I valori riportati sono riferiti al validation set e sono riportati in Tabella 3.7.

#	Epoche	Accuracy	Loss	Rouge1	Rouge2	RougeL
0	1	64.2%	1.692	29.24	13.28	24.26
1	5	82.34%	0.8601	41.45	24.03	35.12
2	20	92.34%	0.4352	56.93	47.50	53.1
3	24	93.49%	0.4315	55.47	47.69	52.69
4	50	94.02%	0.4387	55.20	48.49	52.84

Tabella 3.7: Risultati degli esperimenti su T5-C4 con attention lineare.

T5-C4 con attention quadratica Gli esperimenti sono stati eseguiti con una GPU GeForce RTX 3090 (24GB). I valori riportati sono riferiti al validation set e sono riportati in Tabella 3.8.

#	Epoche	Accuracy	Loss	Rouge1	Rouge2	RougeL
0	100	94.07%	0.4359	55.43	48.62	53.01

Tabella 3.8: Risultati degli esperimenti su T5-C4 con attention quadratica.

3.2.2 BART

Ispirandosi al paper [72], che rappresenta attualmente *SOTA* per il task Data-To-Text, è stato operato un confronto con BART. Vengono importate le librerie necessarie e caricati i dataset:

```
import pandas as pd
from datasets import Dataset, load_metric
from transformers import BartForConditionalGeneration
from transformers import BartTokenizer, Seq2SeqTrainer
from transformers import TrainingArguments

train_data = Dataset.from_pandas(pd.read_csv('train.tsv', sep='\t'))
val_data = Dataset.from_pandas(pd.read_csv('validation.tsv', sep='\t'))
```

Vengono caricati il tokenizer e il modello dall'hub di HuggingFace[2]:

```
model_name = 'facebook/bart-base'
epochs = 50
```

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

```
batch_size = 8

def load_model():
  checkpoints = glob.glob('checkpoint-*')
  if len(checkpoints) > 0:
    model = BartForConditionalGeneration.from_pretrained("./" +
    ↪  checkpoints[0])
    print("Loaded checkpoint from " + checkpoints[0])
  else:
    model = BartForConditionalGeneration.from_pretrained(model_name)
    print("Loaded HuggingFace model: " + model_name)
  return model

tokenizer = BartTokenizer.from_pretrained(model_name)
bart = load_model().to("cuda:1" if torch.cuda.is_available() else "cpu")
```

Vengono selezionate le metriche necessarie alla valutazione sia in fase di evaluation che in fase di testing:

```
rouge = load_metric("rouge")
def compute_metrics(pred):
    labels_ids = pred.label_ids
    pred_ids = pred.predictions
    pred_str = tokenizer.batch_decode(pred_ids, skip_special_tokens=True)
    labels_ids[labels_ids == -100] = tokenizer.pad_token_id
    label_str = tokenizer.batch_decode(labels_ids,
    ↪  skip_special_tokens=True)
    rouge_types = ["rouge1", "rouge2", "rougeLsum"]
    rouge_outputs = rouge.compute(predictions=pred_str,
    ↪  references=label_str, rouge_types=rouge_types)
    rouge1 = rouge_outputs["rouge1"].mid
    rouge2 = rouge_outputs["rouge2"].mid
    rougeLsum = rouge_outputs["rougeLsum"].mid
```

Vengono preparati i dataset di input:

```
train_dataset_model_input = train_dataset.map(
    preprocess_function,
    batched=True,
```

```
    batch_size=batch_size,
    remove_columns=["event", "event_mention"]
)
train_dataset_model_input.set_format(
 type="torch",
 columns=["input_ids", "attention_mask", "labels"]
)
val_dataset_model_input = val_dataset.map(
    preprocess_function,
    batched=True,
    batch_size=batch_size,
    remove_columns=["event", "event_mention"]
)
val_dataset_model_input.set_format(
 type="torch",
 columns=["input_ids", "attention_mask", "labels"]
)
```

Vengono inizializzati gli iperparametri e fatto partire il fine-tuning:

```
args = Seq2SeqTrainingArguments(
    output_dir = "./",
    evaluation_strategy = "steps",
    save_strategy = "steps",
    per_device_train_batch_size = batch_size,
    do_train = True,
    do_eval = True,
    per_device_eval_batch_size = batch_size,
    weight_decay = 0.01,
    num_train_epochs = epochs,
    logging_steps = 3866,
    save_steps = 3866,
    eval_steps = 3866,
    warmup_steps = 1000,
    gradient_accumulation_steps = batch_size // 4,
    eval_accumulation_steps = batch_size // 4,
    predict_with_generate=True,
    fp16=True,
)
trainer = Seq2SeqTrainer(
    model=bart,model,
```

```
    args=args,
    train_dataset=train_dataset_model_input,
    eval_dataset=val_dataset_model_input,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)
```

In particolare, nel codice precedentemente inserito, viene usata la tecnica dell'accumulazione del gradiente, che consente di eseguire fine-tuning su macchine che non dispongono di sufficiente memoria: l'aggiornamento dei pesi viene eseguito quando viene raggiunto un numero di step pari a *gradient_ accumulation_ steps*. Quindi, partendo da un $batch\_ size = 2$, per ottenere un $batch\_ size = 16$ come usato in T5, si deve soddisfare la seguente uguaglianza:

$$total\_ batch\_ size = n\_ gpus \cdot batch\_ size \cdot gradient\_ accumulation\_ steps$$

Sostituendo:

$$total\_ batch\_ size = 1 \cdot 8 \cdot 2 = 16$$

3.2.3 Valutazione

I risultati del confronto (eseguito sul test set), tra T5-C4 (Sezione 3.2.1) e il modello BART proposto in questa sezione, sono riportati in Tabella 3.9, mentre in Tabella 3.10 è presente un esempio di predizione con *T5* e *BART*.

Modello		BART	**T5-base**
Epoche		50	100
Rouge1	Rouge1-Recall	62.31	61.70
	Rouge1-Precision	66.04	67.60
	Rouge1-F1-Score	62.67	**63.09**
Rouge2	Rouge2-Recall	52.48	53.01
	Rouge2-Precision	56.18	58.62
	Rouge2-F1-Score	53.38	**54.72**
RougeL	RougeL-Recall	58.81	64.33
	RougeL-Precision	62.51	58.66
	RougeL-F1-Score	59.38	**60.12**

Tabella 3.9: Risultati del confronto tra T5 e BART sul task di event graph verbalization.

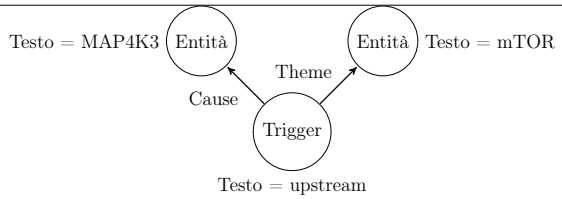
Grafo evento	
Evento codificato	[upstream | Regulation | Speculation = True] [MAP4K3 | Gene_or_gene_product | Cause = upstream] [mTOR | Gene_or_gene_product | Theme = upstream]
Predizione con T5	Indeed, MAP4K3 is required for phosphorylation of known mTOR targets such as S6K1 (S6 kinase 1), and overexpression of MAP4K3 promotes the rapamycin-sensitive phosphoryl.
Predizione con BART	Western blot analysis and phosphorylation analyses showed that MAP4K3 is upstream of mTOR.
Input aumentato	**[MAP4K3 | Gene_or_gene_product | Cause = upstream]** activity is stimulated by administration of amino acids, but not growth factors, and this is insensitive to rapamycin, most likely placing **[MAP4K3 | Gene_or_gene_product | Cause = upstream] [upstream | Regulation | Speculation = True]** of **[mTOR | Gene_or_gene_product | Theme = upstream]**.
Ground-truth	MAP4K3 activity is stimulated by administration of amino acids, but not growth factors, and this is insensitive to rapamycin, most likely placing MAP4K3 upstream of mTOR.

Tabella 3.10: Esempio di un grafo evento, la sua controparte codificata, la predizione di *T5* e di *BART* e il ground-truth. Si noti come *T5* abbia sorprendentemente introdotto l'entità *SK1*, non presente nel *ground-truth*. Ha inoltre aggiunto – oltre a *rapamycin* che non è presente all'interno dell'evento – una sua interazione con *MAP4K3*, che non compare nel *ground-truth*.

3.3 Single-document summarization

3.3.1 T5 with Multi-task learning

È stato effettuato fine-tuning su un modello T5 pre-addestrato su C4, su tre task contemporaneamente: *event-verbalization*, *single-document summarization* e *event-extraction*. I dataset usati sono rispettivamente quelli descritti in Sezione 3.1 e in [16]. Il terzo è il dataset presentato in Sezione 3.1 a cui sono state scambiate colonne di features e output. Il dataset di *single-document summarization* è *Long Summarization* [16] ed è stato ottenuto da *Tensorflow datasets*. Nel caso del training set, il codice di preaprazione del file TSV è il seguente:

```
1 import tensorflow_datasets as tfds
2 ds, info = tfds.load('scientific_papers', shuffle_files=True,
  ↪ with_info=True)
```

Sono stati definiti singolarmente i task:

```
TaskRegistry = dataset_providers.TaskRegistry
TaskRegistry.remove("event_extraction_task")
TaskRegistry.add(
  "event_extraction_task",
  dataset_providers.TextLineTask,
  split_to_filepattern = {
      "train": os.path.join(data_dir, "train_ee.tsv"),
      "validation": os.path.join(data_dir, "validation_ee.tsv"),
  },
  skip_header_lines = 1,
  text_preprocessor = preprocessors.preprocess_tsv,
  metric_fns=[functools.partial(
      rouge_top_beam, beam_size=fine_tuning_cfg.beam_size)]
)
TaskRegistry.remove("event_graph_verbalization_task")
TaskRegistry.add(
  "event_graph_verbalization_task",
  dataset_providers.TextLineTask,
  split_to_filepattern = {
      "train": os.path.join(data_dir, "train.tsv"),
      "validation": os.path.join(data_dir, "validation.tsv"),
  },
  skip_header_lines = 1,
  text_preprocessor = preprocessors.preprocess_tsv,
  metric_fns=[functools.partial(
      rouge_top_beam, beam_size=fine_tuning_cfg.beam_size)]
)
TaskRegistry.remove("summarization_task")
TaskRegistry.add(
  "summarization_task",
  dataset_providers.TextLineTask,
  split_to_filepattern = {
      "train": os.path.join(data_dir, "train_sum.tsv"),
      "validation": os.path.join(data_dir, "validation_sum.tsv")
  },
  skip_header_lines = 1,
  text_preprocessor = preprocessors.preprocess_tsv,
```

```
  metric_fns=[functools.partial(
      rouge_top_beam, beam_size=fine_tuning_cfg.beam_size)]
)
```

Poi è stata definita la mixture di task. Questo fa in modo che il fine-tuning avvenga simultaneamente su tutti i task definiti precedentemente e che T5 ne apprenda la conoscenza all'interno dei suoi pesi:

```
MixtureRegistry = dataset_providers.MixtureRegistry
MixtureRegistry.remove("MTL_mixture")
MixtureRegistry.add(
    "MTL_mixture",
    ["summarization_task", "event_graph_verbalization_task",
    ↪  "event_extraction_task"],
    1.0)
```

Il terzo argomento, posto a 1.0 indica che la proporzione dei dati usati durante il fine-tuning è del 100% per ogni task.

La configurazione base di T5 usata per il task di event graph verbalization (Appendice B) è stata modificato nei seguenti iperparametri:

```
config.steps_per_epoch = 20423
config.num_eval_steps = 831
config.max_input_length = 1024
config.max_target_length = 256
config.max_eval_input_length = 1024
config.max_eval_target_length = 256
```

Esperimenti

Gli esperimenti sono stati eseguiti con 2 GPU GeForce RTX 3090 (48GB in totale). I valori riportati sono riferiti al validation set e sono riportati in Tabella 3.11.

#	**Epoche**	**Accuracy**	**Loss**	**Rouge1**	**Rouge2**	**RougeL**
0	1	0.6914	1.472	18.62	4.934	19.65
1	9	0.7351	1.261	40.31	14.98	36.42

Tabella 3.11: Risultati esperimenti su T5-C4 con configurazione *MTL*.

3.3.2 T5 w/o Multi-task learning

È stato effettuato fine-tuning su un modello T5 pre-addestrato su C4, sul task di single-document summarization. La configurazione custom di T5 usata in Sezione 3.3.1 è stata modificato nei seguenti iperparametri:

```
config.steps_per_epoch = 12689
config.num_eval_steps = 402
```

Esperimenti

Gli esperimenti sono stati eseguiti con 2 GPU GeForce RTX 3090 (48GB in totale). I valori riportati sono riferiti al validation set e sono riportati in Tabella 3.12.

#	**Epoche**	**Accuracy**	**Loss**	**Rouge1**	**Rouge2**	**RougeL**
0	1	60.76%	1.89	18.61	4.95	16.94
1	9	62.52%	1.77	N/A	N/A	N/A

Tabella 3.12: Risultati degli esperimenti su T5-C4 senza configurazione MTL.

Confronto

I risultati del confronto (eseguito sul test set), tra T5-C4 Sezione 3.2.1 e il modello proposto in [16], sono riportati in Tabella 3.13.

Modello		Paper [16]	**T5-base**
Epoche		10	
Rouge1	Rouge1-Recall	35.80	27.24
	Rouge1-Precision		58.11
	Rouge1-F1-Score		**32.97**
Rouge2	Rouge2-Recall	11.05	10.84
	Rouge2-Precision		23.82
	Rouge2-F1-Score		**13.15**
Rouge3	RougeL-Recall	31.80	18.08
	RougeL-Precision		40.29
	RougeL-F1-Score		**22.03**

Tabella 3.13: Risultati del confronto tra il paper [16] e T5.

3.3.3 Valutazione

I risultati del confronto (eseguito sul test set), tra il modello che non presenta configurazione MTL Sezione 3.3.2 e quello che presenta configurazione MTL Sezione 3.3.1, sono riportati in Tabella 3.14.

Modello		T5-base-sum in Sezione 3.3.2	**T5-base-MTL**
Epoche		10	
Rouge1	Rouge1-Recall	27.24	33.45
	Rouge1-Precision	58.11	47.93
	Rouge1-F1-Score	32.97	**36.82**
Rouge2	Rouge2-Recall	10.84	12.10
	Rouge2-Precision	23.82	17.90
	Rouge2-F1-Score	13.15	**13.39**
RougeL	RougeL-Recall	18.08	22.18
	RougeL-Precision	40.29	32.0
	RougeL-F1-Score	22.03	**24.34**

Tabella 3.14: Risultati del confronto tra T5 con [Sezione 3.3.1] e senza [Sezione 3.3.2] configurazione MTL.

In Tabella 3.15 è presente un esempio di predizione di *T5* con e senza configurazione *MTL*.

Output T5 senza MTL	We investigate the bouchaud - mzard(bm) model on a random network. We assume adiabatic and independent assumptions on the stationary distribution function. We derive the equations that determine the stationary distribution function in the non - wealth - condensate phase. We compared our analytical results with those of the numerical simulation , and we found that our theory showed better agreement than did the mean - field theory. We also found that the exponent of the power - law behavior is smaller than that obtained from the mean - field theory.
Output T5 con MTL	We study the bouchaud - mzard model on a random network. We derive the stationary distribution function of wealth in the non - wealth - condensate phase. We find that the stationary distribution function exhibits a power - law behavior, which is a consequence of the wealth - condensation. We also find that the exponent of the pareto s law is smaller than that obtained from mean - field theory.
Ground-truth	We studied the bouchaud - mzard(bm) model, which was introduced to explain pareto s law in a real economy, on a random network using “ adiabatic and independent ” assumptions, we analytically obtained the stationary probability distribution function of wealth. The results shows that wealth - condensation, indicated by the divergence of the variance of wealth, occurs at a larger @xmath0 than that obtained by the mean - field theory, where @xmath0 represents the strength of interaction between agents. We compared our results with numerical simulation results and found that they were in good agreement .

Tabella 3.15: Esempio di una predizione di T5 con e senza configurazione MTL.

Conclusioni e sviluppi futuri

In questa tesi è stato esplorato il task di event graph verbalization in ambito biomedico. È stato proposto un dataset originale, dato dall'esigenza non solo di associare al linguaggio naturale un evento, ma anche di arricchire quest'ultimo attraverso modificatori per sfruttare il meccanismo di *controllable verbalization*, che aggiunge alla verbalizzazione la possibilità di scelta dello stile.

Il task è stato affrontato implementando un modello *T5* in *Flax* (*T5X*), usandone due configurazioni: la prima in combinazione con una "classica" *attention* quadratica, la seconda con una più innovativa *attention* lineare. Questo ci ha permesso di evincere un miglioramento risibile nei tempi di training, che diverrebbe certamente più marcato nel caso di un dataset dalle proporzioni maggiori. Avendo tracciato una baseline con BART, si traggono le conclusioni che T5 possiede prestazioni comparabili in termini di Rouge-1-2-L e superiori in termini qualitativi. Si è dimostrato che l'utilizzo di informazione strutturata come gli eventi permette di conferire un *boost* prestazionale alla risoluzione di altri task *NLP*, come *summarization* sul dataset *long summarization* [16]. In particolare è stato notato un aumento di circa 3.85 punti su Rouge1, di 0.24 su Rouge2 e di 2.31 punti su RougeL. In conclusione, gli eventi rappresentano un tassello importante anche per task *NLP* non direttamente correlati ad essi e la ricerca futura potrebbe rappresentare un'ulteriore conferma in tal senso.

In futuro si potrebbero sfruttare le predizioni di *T5* per eseguire nuovamente un task di *event-extraction* al fine di migliorarne le performance. Più in generale, il dataset potrà essere sfruttato per inserire un modulo che sfrutti la verbalizzazione di grafi in un'architettura più ampia, i.e. in coda ad un'architettura *end-to-end* che estragga gli eventi, li aggreghi per similarità semantica e li usi per eseguire *summarization*. Inoltre si potrebbe usare questo dataset come base per risolvere un task di *information retrieval* (*IR*) ispirato al lavoro di *DeepMind* [73]. Un primo approccio potrebbe essere quello di calcolare, per ogni possibile coppia testo - grafo evento, uno score di similarità, espresso come probabilità. Essendo la complessità computazionale $\mathcal{O}(n^2)$ questo approccio non è scalabile per quantità elevate di dati. Una seconda soluzione potrebbe consistere nell'usare modelli basati su *deep metric learning* [74], in grado di apprendere una funzione $\phi : \mathbb{R}^n \mapsto \mathbb{R}^d$ $(d \gg n)$ in grado di mappare elementi

multi-modali (in questo caso grafi-evento e testo) in uno spazio vettoriale latente $\mathbb{R}^d$, minimizzando la distanza euclidea tra i punti all'aumentare della similarità tra essi. La funzione obiettivo su cui il modello dovrebbe essere addestrato necessita di un input composto da:

1. una *query* (ancora), nel caso di questa tesi un testo in linguaggio naturale rappresentato da un *event mention*;
2. un esempio "positivo", cioè l'evento corrispondente alla *query*;
3. uno o più esempi "negativi", cioè eventi completamente scollegati – di tipo o provenienza diversi – dalla *query* e da cui quest'ultima dovrà distanziarsi il più possibile.

Questo potrebbe portare alla creazione di diverse applicazioni pratiche. Una fra tutte consentirebbe a un medico di accedere alla letteratura scientifica in modo agile, i.e. evitando di leggere tutti i paper di interesse su un argomento: dato un testo, vengono restituite le k risposte più congrue alla *query* in input. (i.e. "Cancer stem cells tumor progression").

Appendice A

Docker

Di seguito il *Dockerfile* usato per la creazione dell'immagine docker.

```
FROM nvcr.io/nvidia/pytorch:20.09-py3
LABEL maintainer="DISI NLU Research Group"

# Zero interaction (default answers to all questions)
ENV DEBIAN_FRONTEND=noninteractive

# Set work directory
WORKDIR /event_graph_verbalization/

# Add requirements file
ADD requirements.txt /event_graph_verbalization/

# Install dependencies
RUN apt-get update -y && \
    apt-get install -y curl \
                       git \
                       bash \
                       nano \
                               ssmtp \
                               cron \
                               lsof \
                       subversion && \
    apt-get autoremove -y && \
    apt-get clean -y && \
    rm -rf /var/lib/apt/lists/*
RUN pip install --upgrade pip
```

```
RUN pip install wrapt --upgrade --ignore-installed
RUN pip install -r requirements.txt
RUN pip install --upgrade jax jaxlib==0.1.57+cuda110 -f
↪  https://storage.googleapis.com/jax-releases/jax_releases.html
RUN pip install jax==0.2.8
RUN pip uninstall tensorboard-plugin-dlprof
RUN pip uninstall tensorboard==2.4.0
RUN pip install tensorboard==2.6.0
RUN pip install gdown
RUN pip install rouge_score
RUN conda install gh --channel conda-forge
ENTRYPOINT gh auth login

# Setup SSMTP
RUN echo "DISI=USERNAME@gmail.com" >> /etc/ssmtp/ssmtp.conf
RUN echo "mailhub=smtp.gmail.com:465" >> /etc/ssmtp/ssmtp.conf
RUN echo "AuthUser=USERNAME@gmail.com" >> /etc/ssmtp/ssmtp.conf
RUN echo "AuthPass=PASSWORD" >> /etc/ssmtp/ssmtp.conf
RUN echo "UseTLS=YES" >> /etc/ssmtp/ssmtp.conf

# Back to default frontend
ENV DEBIAN_FRONTEND=dialog
```

Il file di requirements usato da *Dockerfile*:

```
datasets==1.2.1
dm-tree==0.1.5
flax==0.3.0
gin-config==0.4.0
lazyprofiler==0.1.1
matplotlib==3.3.2
ml-collections==0.1.0
numpy==1.19.2
nvidia_smi==0.1.3
nvidia_ml_py3==7.352.0
t5==0.7.1
tensorboard==2.4.0
tensorflow==2.4.0
tensorflow-datasets==4.2.0
tensorflow-text==2.4.3
tqdm==4.49.0
```

```
transformers==4.2.0
datasets==1.0.2
git-python==1.0.3
sacrebleu==1.4.12
```

L'immagine è stata creata con il seguente comando:

```
docker build -t egv-image .
```

Il container è stato creato con il seguente comando:

```
docker run --name egv-container -v LOCAL_FOLDER:/CONTAINER_FOLDER --gpus
↪  device=0 -m 20g -p 0.0.0.0:37339:8888 -it egv-image /bin/bash
```

In particolare:

- –name: indica il del container da creare;
- -v: mappa una cartella sulla macchina fisica con una sul container;
- –gpus: indica le GPU visibili dal container (all per includerle tutte);
- -m: indica la massima quantità di memoria GPU visibile dal container;
- 0.0.0.0:37339:8888: mappa la porta 8888 del container (generalmente usata da *Jupyter notebook*) alla 37339 sulla macchina fisica;
- -it: indica il nome dell'immagine da usare;
- /bin/bash: indica la shell da usare;

Alla macchina remota ci si connette tramite SSH, mappando la porta 37339 remota ad una porta locale attualmente libera:

```
ssh -L LOCAL_PORT:127.0.0.1:REMOTE_PORT USERNAME@REMOTE_IP -p SERVER_PORT
```

Per scaricare i checkpoint, non avendo la possibilità di salvarli tramite GLF (Git Large File Storage), è sato usato il seguento comando:

```
scp USERNNAME@REMOTE_IP:/path/to/checkpoint/checkpoint_CHECKPOINT_OFFSET ./
```

Appendice B

Configurazione di T5

Di seguito il file di configurazione usato per gli esperimenti di fine-tuning su un modello T5X, pre-addestrato su C4. Negli esperimenti cambiano il numero di epoche e l'offset del checkpoint di partenza. Si noti inoltre la presenza dell'attention lineare (*fast_relu_attention_fn*), del caricamento dei pesi iniziali dai servizi di *Google Cloud Storage* e dell'offset dell'ultimo checkpoint.

```
import ml_collections
from linear_attention.linear_attentions import get_performer_attentions
fast_softmax_attention_fn, fast_relu_attention_fn =
↪  get_performer_attentions()
config = ml_collections.ConfigDict()
# We load weights from authors' checkpoint on C4 dataset (no PyTorch
↪  pre-trained models)
config.load_pytorch_weights = None
# T5 pretrained checkpoint to use.
config.restore_t5_checkpoint = (
  'gs://t5-data/pretrained_models/base/model.ckpt-999900')
# Name of T5 task/mixture to use for finetuning.
config.mixture_or_task_name = 'event_graph_verbalization_task'
# Whether to use T5 preprocessing cache for train task/mixture.
config.train_use_cached = False
# Name of T5 task/mixture to use for evaluation.
config.eval_mixture_or_task_name = 'event_graph_verbalization_task'
# Whether to use T5 preprocessing cache for eval task/mixture.
config.eval_use_cached = False
# Name of T5 task/mixture split to use for evaluation.
config.eval_split = 'validation'
# Whether to save model checkpoints.
```

```
config.save_checkpoints = True
# Whether to restore from existing model checkpoints.
config.restore_checkpoints = True
# Save a checkpoint every Nth epoch.
config.checkpoint_freq = 1
# Number of epochs to train for.
config.num_epochs = NUM_EPOCHS
# Number of steps per epoch (i.e., number of batches for each epoch)
# To see the entire dataset, it should be: number of records / batch_size
# i.e., 61869 / 16
config.steps_per_epoch = 3866
# Number of steps to take during evaluation.
# i.e., 3437 / 16
config.num_eval_steps = 214
# Collect Xprof traces on host 0.
config.xprof = True
# Whether to use hardware rng for dropout.
config.hardware_rng = True
# Integer for PRNG random seed.
config.random_seed = 0
# Use infeed in training loop.
config.infeed = True
# Total batch size for training.
config.batch_size = 16
# Total batch size for inference on tasks.
config.eval_batch_size = 16
# Number of gradient-accumulating microbatches.
config.microbatches = 2
# Number of SPMD partitions to use.
config.num_partitions = 1
# Beam size for inference.
config.beam_size = 4
# Learning rate schedule.
config.schedule = 'constant'
# Base learning rate.
config.learning_rate = 0.001
# Linear learning rate warmup.
config.warmup_steps = 1000
# Cross entropy loss label smoothing.
config.label_smoothing = 0.0
# Cross entropy auxilliary z-loss coefficient.
```

```
config.z_loss = 0.0001
# Starting step offset of fine-tuning phase for Adafactor.
config.step_offset = CHECKPOINT_OFFSET
# Maximum length cutoff for training examples.
config.max_input_length = 512
config.max_target_length = 62
# Maximum length cutoff for eval examples.
config.max_eval_input_length = 512
config.max_eval_target_length = 62
# Vocabulary size if `vocab_path` is not given.
config.vocab_size = 32128
# Inputs and targets share embedding.
config.share_embeddings = True
# Final logit transform uses embedding matrix transpose.
config.logits_via_embedding = True
# Number of transformer layers.
config.num_layers = 12
# Size of query/key/value for attention.
config.qkv_dim = 768
# Size of embeddings.
config.emb_dim = 768
# Size of the MLP.
config.mlp_dim = 3072
# Activations in MLP input.
config.mlp_activations = ('relu',)
# Number of attention heads.
config.num_heads = 12
# Number of relative-attention bins.
config.relative_attention_num_buckets = 32
# Number of relative-attention bins.
config.relative_attention_max_distance = 128
# Dropout rate.
config.dropout_rate = 0.1
# Attention dropout rate.
config.attention_dropout_rate = 0.1
# Use bfloat16 mixed precision training instead of float32.
config.use_bfloat16 = True
# --> MODULAR ATTENTION <--
config.attention_fn = fast_relu_attention_fn
config.infeed = False
config.final_dense_layer = False
```

Bibliografia

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

[2] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[3] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[4] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[5] Jena D Hwang, Chandra Bhagavatula, Ronan Le Bras, Jeff Da, Keisuke Sakaguchi, Antoine Bosselut, and Yejin Choi. Comet-atomic 2020: On symbolic and neural commonsense knowledge graphs. *arXiv preprint arXiv:2010.05953*, 2020.

[6] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*, 2017.

[7] Bill Yuchen Lin, Seyeon Lee, Rahul Khanna, and Xiang Ren. Birds have four legs?! numersense: Probing numerical commonsense knowledge of pre-trained language models. *arXiv preprint arXiv:2005.00683*, 2020.

[8] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big?. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 610–623, 2021.

[9] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: a survey on explainable artificial intelligence (xai). *IEEE access*, 6:52138–52160, 2018.

[10] Robert L Logan IV, Nelson F Liu, Matthew E Peters, Matt Gardner, and Sameer Singh. Barack's wife hillary: Using knowledge-graphs for fact-aware language modeling. *arXiv preprint arXiv:1906.07241*, 2019.

[11] Oshin Agarwal, Heming Ge, Siamak Shakeri, and Rami Al-Rfou. Knowledge graph based synthetic corpus generation for knowledge-enhanced language model pre-training. *arXiv preprint arXiv:2010.12688*, 2020.

[12] Esther Landhuis. Scientific literature: Information overload. *Nature*, 535(7612):457–458, 2016.

[13] Xu Zou, Da Yin, Qingyang Zhong, Hongxia Yang, Zhilin Yang, and Jie Tang. Controllable generation from pre-trained language models via inverse prompting. *arXiv preprint arXiv:2103.10685*, 2021.

[14] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2020.

[15] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[16] Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. A discourse-aware attention model for abstractive summarization of long documents. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 2018.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training (2018), 2018.

[20] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[22] Wei Xiang and Bang Wang. A survey of event extraction from text. *IEEE Access*, 7:173111–173137, 2019.

[23] Hai-Long Trieu, Thy Thy Tran, Khoa NA Duong, Anh Nguyen, Makoto Miwa, and Sophia Ananiadou. Deepeventmine: end-to-end neural nested event extraction from biomedical texts. *Bioinformatics*, 36(19):4910–4917, 2020.

[24] Makoto Miwa and Sophia Ananiadou. NaCTeM EventMine for BioNLP 2013 CG and PC tasks. In Claire Nédellec, Robert Bossy, Jin-Dong Kim, Jung-Jae Kim, Tomoko Ohta, Sampo Pyysalo, and Pierre Zweigenbaum, editors, *Proceedings of the BioNLP Shared Task 2013 Workshop, Sofia, Bulgaria, August 9, 2013*, pages 94–98. Association for Computational Linguistics, 2013.

[25] Jari Björne and Tapio Salakoski. TEES 2.2: biomedical event extraction for diverse corpora. *BMC bioinformatics*, 16(16):1–20, 2015.

[26] Lishuang Li, Shanshan Liu, Meiyue Qin, Yiwen Wang, and Degen Huang. Extracting Biomedical Event with Dual Decomposition Integrating Word Embeddings. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 13(4):669–677, 2016.

[27] Weizhong Zhao, Jinyong Zhang, Jincai Yang, Tingting He, Huifang Ma, and Zhixin Li. A novel joint biomedical event extraction framework via two-level modeling of documents. *Information Sciences*, 550:27–40, 2021.

[28] Sabenabanu Abdulkadhar, Balu Bhasuran, and Jeyakumar Natarajan. Multiscale laplacian graph kernel combined with lexico-syntactic patterns for biomedical event extraction from literature. *Knowledge and Information Systems*, 63(1):143–173, 2021.

[29] Arantza Casillas, Arantza Díaz De Ilarraza, Koldo Gojenola, Maite Oronoz, and German Rigau. Using kybots for extracting events in biomedical texts. In *Proceedings of BioNLP Shared Task 2011 Workshop*, pages 138–142, 2011.

[30] Halil Kilicoglu and Sabine Bergler. Syntactic dependency based heuristics for biological event extraction. In *Proceedings of the BioNLP 2009 Workshop Companion Volume for Shared Task*, pages 119–127, 2009.

[31] Tsendsuren Munkhdalai, Oyun-Erdene Namsrai, and Keun Ho Ryu. Self-training in significance space of support vectors for imbalanced biomedical event data. *BMC bioinformatics*, 16(7):1–8, 2015.

[32] Sudha Rao, Daniel Marcu, Kevin Knight, and Hal Daumé III. Biomedical event extraction using abstract meaning representation. In *BioNLP 2017*, pages 126–135, 2017.

[33] Xing David Wang, Leon Weber, and Ulf Leser. Biomedical event extraction as multi-turn question answering. In *Proceedings of the 11th International Workshop on Health Text Mining and Information Analysis*, pages 88–96, 2020.

[34] Lifu Huang, Taylor Cassidy, Xiaocheng Feng, Heng Ji, Clare Voss, Jiawei Han, and Avirup Sil. Liberal event extraction and event schema induction. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 258–268, 2016.

[35] Yubo Chen, Liheng Xu, Kang Liu, Daojian Zeng, and Jun Zhao. Event extraction via dynamic multi-pooling convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 167–176, Beijing, China, July 2015. Association for Computational Linguistics.

[36] Jari Björne and Tapio Salakoski. Biomedical event extraction using convolutional neural networks and dependency parsing. In *Proceedings of the BioNLP 2018 workshop*, pages 98–108, 2018.

[37] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha,*

Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, pages 1724–1734. ACL, 2014.

[38] Trung Minh Nguyen and Thien Huu Nguyen. One for all: Neural joint modeling of entities and events. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 6851–6858. AAAI Press, 2019.

[39] Diya Li, Lifu Huang, Heng Ji, and Jiawei Han. Biomedical Event Extraction based on Knowledge-driven Tree-LSTM. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 1421–1430. Association for Computational Linguistics, 2019.

[40] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676*, 2019.

[41] Jesse Dodge, Maarten Sap, Ana Marasovic, William Agnew, Gabriel Ilharco, Dirk Groeneveld, and Matt Gardner. Documenting the english colossal clean crawled corpus. *arXiv preprint arXiv:2104.08758*, 2021.

[42] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[43] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *arXiv preprint arXiv:1905.00537*, 2019.

[44] Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*, 2016.

[45] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[46] Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, et al. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the ninth workshop on statistical machine translation*, pages 12–58, 2014.

[47] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

[48] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[49] William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

[50] Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

[51] Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2012.

[52] Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*, 2011.

[53] Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 252–262, 2018.

[54] Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. Record: Bridging the gap between human and machine commonsense reading comprehension. *arXiv preprint arXiv:1810.12885*, 2018.

[55] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

[56] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[57] Mohammad Golam Sohrab and Makoto Miwa. Deep exhaustive model for nested named entity recognition. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2843–2849, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[58] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[59] Giovanni Paolini, Ben Athiwaratkun, Jason Krone, Jie Ma, Alessandro Achille, Rishita Anubhai, Cicero Nogueira dos Santos, Bing Xiang, and Stefano Soatto. Structured prediction as translation between augmented natural languages. *arXiv preprint arXiv:2101.05779*, 2021.

[60] Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. Creating training corpora for NLG micro-planners. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 179–188, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[61] Jin-Dong Kim, Tomoko Ohta, Kanae Oda, and Jun'ichi Tsujii. From text to pathway: corpus annotation for knowledge acquisition from biomedical literature. In *Proceedings of The 6th Asia-Pacific Bioinformatics Conference*, pages 165–175. World Scientific, 2008.

[62] Jin-Dong Kim, Ngan Nguyen, Yue Wang, Jun'ichi Tsujii, Toshihisa Takagi, and Akinori Yonezawa. The genia event and protein coreference tasks of the bionlp shared task 2011. In *BMC bioinformatics*, volume 13, pages 1–12. BioMed Central, 2012.

[63] Tomoko Ohta, Sampo Pyysalo, and Jun'ichi Tsujii. Overview of the epigenetics and post-translational modifications (epi) task of bionlp shared task 2011. In *Proceedings of BioNLP Shared Task 2011 Workshop*, pages 16–25, 2011.

[64] Sampo Pyysalo, Tomoko Ohta, Rafal Rak, Dan Sullivan, Chunhong Mao, Chunxia Wang, Bruno Sobral, Jun'ichi Tsujii, and Sophia Ananiadou. Overview of the infectious diseases (id) task of bionlp shared task 2011. In *Proceedings of BioNLP Shared Task 2011 Workshop*, pages 26–35, 2011.

[65] Sampo Pyysalo, Tomoko Ohta, Makoto Miwa, Han-Cheol Cho, Jun'ichi Tsujii, and Sophia Ananiadou. Event extraction across multiple levels of biological organization. *Bioinformatics*, 28(18):i575–i581, 2012.

[66] Makoto Miwa, Paul Thompson, John McNaught, Douglas B Kell, and Sophia Ananiadou. Extracting semantically enriched events from biomedical literature. *BMC bioinformatics*, 13(1):1–24, 2012.

[67] Jin-Dong Kim, Yue Wang, and Yamamoto Yasunori. The genia event extraction shared task, 2013 edition-overview. In *Proceedings of the BioNLP Shared Task 2013 Workshop*, pages 8–15, 2013.

[68] Sampo Pyysalo, Tomoko Ohta, and Sophia Ananiadou. Overview of the cancer genetics (cg) task of bionlp shared task 2013. In *Proceedings of the BioNLP Shared Task 2013 Workshop*, pages 58–66, 2013.

[69] Tomoko Ohta, Sampo Pyysalo, Rafal Rak, Andrew Rowley, Hong-Woo Chun, Sung-Jae Jung, Sung-Pil Choi, Sophia Ananiadou, and Jun'ichi Tsujii. Overview of the pathway curation (pc) task of bionlp shared task 2013. In *Proceedings of the BioNLP Shared Task 2013 Workshop*, pages 67–75, 2013.

[70] Jung-Jae Kim, Xu Han, Vivian K Lee, and Dietrich Rebholz Schuhmann. Gro task: Populating the gene regulation ontology with events and relations. In *Proceedings of the BioNLP Shared Task 2013 Workshop*, pages 50–57, 2013.

[71] Susan H Rodger and Thomas W Finley. *JFLAP: an interactive formal languages and automata package*. Jones & Bartlett Learning, 2006.

[72] Qingyun Wang, Semih Yavuz, Xi Victoria Lin, Heng Ji, and Nazneen Rajani. Stage-wise fine-tuning for graph-to-text generation. In *Proceedings of the ACL-IJCNLP 2021 Student Research Workshop*, pages 16–22, Online, August 2021. Association for Computational Linguistics.

[73] Luyu Wang, Yujia Li, Ozlem Aslan, and Oriol Vinyals. Wikigraphs: A wikipedia text-knowledge graph paired dataset. In *Proceedings of the Graph-Based Methods for Natural Language Processing (TextGraphs)*, pages 67–82, 2021.

[74] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In Aasa Feragen, Marcello Pelillo, and Marco Loog, editors, *Similarity-Based Pattern Recognition*, pages 84–92, Cham, 2015. Springer International Publishing.

Ringraziamenti

Prima di tutto vorrei ringraziare il mio relatore, il professor Gianluca Moro per avermi dato l'opportunità di lavorare assieme a lui e per aver creduto nelle mie capacità. Ringrazio di cuore anche il mio co-relatore, il dottor Giacomo Frisoni, per avermi costantemente aiutato, sia dal punto di vista accademico che sopratutto umano, nel raggiungere l'obiettivo finale. Alla mia ragazza Laura, a tutta la mia famiglia e ai miei amici più cari va un ringraziamento speciale per avermi consigliato, supportato e spronato. Vorrei citare anche i miei compagni di corso e – in modo particolare – Davide e Veronika, il cui aiuto e generosità sono stati a dir poco fondamentali per arrivare alla fine di questo percorso. Infine vorrei ringraziare i professori incontrati durante il corso di studi, i cui preziosi insegnamenti mi hanno permesso di arrivare fino a questo punto.