ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN
INGEGNERIA INFORMATICA

INTEGRAZIONE DI RETI NEURALI CON APPROCCI SIMBOLICI PER PROBLEMI DI RAGIONAMENTO DEDUTTIVO IN LINGUAGGIO NATURALE

TESI DI LAUREA IN
FONDAMENTI DI INTELLIGENZA ARTIFICIALE

CANDIDATO:
Giacomo Fantazzini

RELATORE:
Prof. Federico Chesani

CORRELATORE:
Prof.ssa Paola Mello

Anno Accademico 2021/2022

Abstract

Uno degli obiettivi più ambizioni e interessanti dell'informatica, specialmente nel campo dell'intelligenza artificiale, consiste nel raggiungere la capacità di far ragionare un computer in modo simile a come farebbe un essere umano. I più recenti successi nell'ambito delle reti neurali profonde, specialmente nel campo dell'elaborazione del testo in linguaggio naturale, hanno incentivato lo studio di nuove tecniche per affrontare tale problema, a cominciare dal ragionamento deduttivo, la forma più semplice e lineare di ragionamento logico.

La domanda fondamentale alla base di questa tesi è infatti la seguente: in che modo una rete neurale basata sull'architettura *Transformer* [7] può essere impiegata per avanzare lo stato dell'arte nell'ambito del ragionamento deduttivo in linguaggio naturale?

Nella prima parte di questo lavoro presento uno studio approfondito di alcune tecnologie recenti che hanno affrontato questo problema con intuizioni vincenti. Da questa analisi emerge come particolarmente efficace l'integrazione delle reti neurali con tecniche simboliche più tradizionali.

Nella seconda parte propongo un focus sull'architettura *ProofWriter* [17], che ha il pregio di essere relativamente semplice e intuitiva pur presentando prestazioni in linea con quelle dei concorrenti. Questo approfondimento mette in luce la capacità dei modelli *T5* [12], con il supporto del framework *HuggingFace* [35], di produrre più risposte alternative, tra cui è poi possibile cercare esternamente quella corretta.

Nella terza e ultima parte fornisco un prototipo che mostra come si può impiegare tale tecnica per arricchire i sistemi tipo *ProofWriter* con approcci simbolici basati su nozioni linguistiche, conoscenze specifiche sul dominio applicativo o semplice buonsenso.

Ciò che ne risulta è un significativo miglioramento dell'accuratezza rispetto al *ProofWriter* originale, ma soprattutto la dimostrazione che è possibile sfruttare tale capacità dei modelli *T5* per migliorarne le prestazioni.

Indice

1. Introduzione

L'obiettivo principale di questa tesi consiste nel proporre uno studio sugli sviluppi più recenti nell'ambito del ragionamento automatizzato in campo informatico, arricchito da una mia proposta sulla direzione che potrebbe prendere la ricerca per avanzare ulteriormente lo stato dell'arte.

In questo ambito, è oggi particolare oggetto di interesse l'intuizione di sfruttare tecniche sub-simboliche per superare i limiti tipici delle tecniche simboliche, cercando però di non rinunciare ai vantaggi di queste ultime. Vedremo che, al momento, le soluzioni più promettenti propongono un'integrazione dei due paradigmi e si concentrano specialmente sull'implementazione del ragionamento deduttivo, il meccanismo alla base della logica classica.

Gli esperimenti che ho condotto, e che riporto nei prossimi capitoli, sono stati svolti in cloud tramite un abbonamento Pro alla piattaforma *Google Colab*. Non è quindi possibile avere una descrizione completa e costante dell'hardware utilizzato, riporto però che tale setup mi ha tipicamente assegnato più di 12 GiB di RAM e una GPU Nvidia Tesla P100.

Nella sezione **Automatizzare il ragionamento deduttivo** si inquadrerà a grandi linee il contesto storico del problema, mentre nella sezione **Il Transformer e le architetture derivate** si introdurrà l'omonima architettura, che è tuttora al centro della sperimentazione nell'ambito dell'elaborazione dei testi in linguaggio naturale. Nella sezione **Definizione del problema** si descriverà dettagliatamente il problema oggetto della tesi.

1.1. Automatizzare il ragionamento deduttivo

Il ragionamento deduttivo trova le sue radici ai tempi dell'Antica Grecia. In particolare, in Occidente si sviluppa a partire dalla teoria dei sillogismi di Aristotele, che studia quando due premesse vere consentono di giungere ad una conclusione altrettanto vera (*Tutti gli uomini sono mortali. Socrate è un uomo. Dunque Socrate è mortale*). Dai sillogismi nascono la logica

dei predicati e, si potrebbe dire, l'intera logica matematica. L'interesse di natura filosofica per queste discipline è rimasto vivo fino ai giorni nostri, quando con l'avvento e la diffusione dei computer è migrato nel contesto della programmazione.

Nell'ambito dell'intelligenza artificiale, il primo contributo al ragionamento in linguaggio formale si deve probabilmente all'articolo del 1959 *Programs with Common Sense* [1]. Esso propone il concetto di *AdviceTaker*, un ipotetico programma che manipolerebbe il linguaggio deducendo conclusioni a partire da una certa base di conoscenza. Descrivendo solamente l'ambiente e il compito da svolgere, il programma deve produrre autonomamente un piano di azioni da compiere per raggiungere l'obiettivo. Procedure ed euristiche vengono descritte nello stesso linguaggio e il sistema deve essere in grado di accettare nuova conoscenza e di adattarsi ad essa senza venire esplicitamente riprogrammato.

Come riportato nella sezione storica di *Artificial Intelligence: A Modern Approach* [2], gli anni successivi sono stati segnati da importanti successi nell'ambito della pianificazione automatica e nello sviluppo di reti neurali e sistemi esperti, che avrebbero successivamente influenzato positivamente lo studio del ragionamento automatico, ma anche da significativi fallimenti, che hanno rallentato molto la diffusione delle tecniche di intelligenza artificiale rispetto alle stime iniziali.

Tra i sistemi basati sulla programmazione logica, è rilevante citare il programma *Talk* [3], proposto nel 1987 e in grado di interpretare semplici frasi in inglese e convertirle in linguaggio Prolog in modo da poterle usare per rispondere a domande successivamente poste dall'utente. Il sistema lavora analizzando una frase, costruendone una rappresentazione in forma logica, convertendola in una clausola di Horn e aggiungendola alla base di dati Prolog. Se invece la frase è interpretata come domanda, il programma calcola la risposta e la restituisce. Il sistema fornisce però solo risposte poco generali ed è limitato ad un vocabolario ridotto e precisamente definito.

La principale limitazione di questi sistemi resta tutt'oggi la necessità di rappresentare la conoscenza in un linguaggio formale e non in uno naturale, come potrebbe essere l'inglese comunemente parlato, e dunque la necessità di coinvolgere tecnici ed esperti del dominio per ciascuna applicazione reale. Esiste anche la possibilità di automatizzare l'analisi sintattica e semantica delle frasi per poi procedere alla traduzione automatica da linguaggio naturale a linguaggio formale, tuttavia non è affatto banale ideare regole sufficientemente generali ed in pratica ciascuna applicazione richiede tipicamente un proprio dizionario con regole specifiche, si prendano a titolo di esempio le architetture *NLProlog* [4] e *PWM-PWL* [5], successivamente trattate nel dettaglio.

Un'interessante proposta che unisce il linguaggio naturale a quello formale è rappresentata dal *Logical English* [6], un linguaggio di programmazione dichiarativo con regole formali, ma impostato in modo da poter essere letto senza problemi anche da chi non s'intende di programmazione. Pur essendo un linguaggio general-purpose, ha visto al momento applicazioni quasi esclusivamente in ambito legale, tanto da poter essere considerato un Domain-Specific Language (DSL).

Visto il recente successo delle tecniche di deep-learning, che hanno permesso l'avanzamento dello stato dell'arte in tantissimi campi di ricerca, incluso il Natural Language Processing (NLP), vale certamente la pena di investigare la possibilità di delegare l'analisi sintattico-semantica o addirittura l'intero processo ad una rete neurale. È tuttavia necessario tenere presente che impiegando tecniche sub-simboliche non è scontato ottenere una dimostrazione del ragionamento svolto insieme alla risposta, mentre nei sistemi simbolici ciò risulta tipicamente triviale.

1.2. Il Transformer e le architetture derivate

Il principale avanzamento tecnologico degli ultimi anni in ambito NLP è rappresentato dall'articolo del 2017 *Attention Is All You Need* [7], in cui viene presentata la rete neurale *Transformer*. Basata sul meccanismo di

self-attention, essa consente un'ottima parallelizzazione delle operazioni su GPU e permette addestramenti molto veloci.

Il precedente stato dell'arte era, al contrario, rappresentato da modelli estremamente sequenziali basati su Long Short-Term Memory (LSTM) e Recurrent Neural Network (RNN) e in alcuni casi Convolutional Neural Network (CNN).

Il *Transformer* originale è stato ideato per compiere traduzioni di testo da una lingua all'altra ed è composto da due blocchi principali, chiamati encoder e decoder. L'encoder ha il compito di interpretare correttamente il significato del testo in input generando una rappresentazione del contesto, mentre il decoder deve apprendere come costruire l'output corretto a partire da tale contesto.

Da questo studio sono scaturite varie architetture di successo, come *BERT* (Bidirectional Encoder Representations from Transformers) [8], basata sul solo encoder e specializzata nell'interpretazione del testo, e *GPT* (Generative Pre-Training) [9], basata sul solo decoder e specializzata nella generazione creativa. Questi modelli hanno ispirato a loro volta nuove varianti sempre più performanti, pre-addestrate e appositamente pensate per applicazioni di transfer-learning. Il tipico flusso di lavoro consiste quindi nel selezionare una rete già addestrata a trattare il linguaggio naturale e sottoporla a un addestramento addizionale mirato per il compito specifico da apprendere.

Tra le varianti di *BERT* si distingue certamente *RoBERTa* (Robustly Optimized BERT Pretraining Approach) [10], un modello molto popolare e performante ottenuto ricreando lo studio originale, ritarando svariati parametri e sottoponendo il modello ad un addestramento più lungo e intenso. *ALBERT* (A Lite BERT) [11] è invece una versione particolarmente ottimizzata di *BERT*, in gran parte grazie alla condivisione dei parametri tra livelli diversi. Ciò comporta un'enorme riduzione del numero di para-

metri a fronte di una minima perdita in precisione, compensabile successivamente dalla possibilità di ingrandire il modello mantenendo inalterato il carico computazionale.

Le più nuove e promettenti architetture derivanti dal *Transformer* sono quelle della famiglia *T5* (Text-to-Text Transfer Transformer) [12], costituite da uno stack di encoder seguito da uno stack di decoder. L'obiettivo è fornire un unico framework con input e output in forma testuale in grado di svolgere con precisione i compiti richiesti senza alterare l'architettura della rete, ma affidandosi unicamente al fine-tuning.

Lo studio *WT5* (Why, T5?) [13] mostra risultati promettenti nella capacità di addestrare i modelli *T5* a fornire una spiegazione assieme al proprio output. Questi modelli si dimostrano effettivamente in grado di trasferire le capacità esplicative da un task all'altro, tuttavia non vi è garanzia che la spiegazione fornita sia quella coerentemente creduta dal modello e non una spiegazione di facciata più o meno convincente.

D'altra parte, lo studio *Reasoning with Transformer-based Models* [14] evidenzia come le reti basate sul *Transformer* sembrino spesso apprendere e replicare pattern statistici anziché compiere ragionamenti logici. Viene inoltre evidenziato un importante limite teorico di queste architetture, già esplorato da altri studi come *Theoretical Limitations of Self-Attention in Neural Sequence Models* [15] e *On the Practical Ability of RNNs to Recognize Hierarchical Languages* [16], ovvero l'incapacità di emulare il funzionamento dei pushdown automata. Viene dunque dimostrata una capacità espressiva inferiore rispetto alle architetture precedenti basate su LSTM e RNN e vengono riportati alcuni esempi che ne mostrano il relativo impatto su alcune concrete applicazioni di NLP.

Nonostante i dubbi legittimi sulla validità generale di questa tecnologia, sono innegabili gli ottimi risultati ottenuti nelle sue applicazioni concrete. Vale dunque la pena di chiedersi se e in che modo i *Transformer* possano essere impiegati per avanzare lo stato dell'arte anche nel campo del ragionamento automatizzato in linguaggio naturale.

1.3. Definizione del problema

L'obiettivo ultimo, aperto e generale, consisterebbe nel predisporre un sistema che (1) accetti in input domande e situazioni descritte in un linguaggio naturale, (2) ragioni logicamente su quanto ricevuto e (3) produca in output, assieme a ciascuna risposta, una dimostrazione formale dei ragionamenti che lo hanno condotto a tali conclusioni.

Il problema qui effettivamente trattato consiste in un rilassamento di questi vincoli, in linea con l'attuale stato della tecnologia, ma comunque sufficientemente sfidante. Parliamo dunque di sistemi che siano in grado di (1) accettare input, e preferibilmente fornire output, in linguaggio naturale, (2) compiere una qualche forma di ragionamento logico, preferibilmente deduttivo e (3) fornire una spiegazione del ragionamento svolto, preferibilmente una vera e propria dimostrazione.

Sono speciale oggetto di interesse, in questo ambito, quelle soluzioni che includono nella propria architettura reti neurali profonde derivanti dal *Transformer* [7]. Una volta poste le basi per comprendere le principali criticità del problema e l'attuale stato della ricerca, infatti, il focus si sposta sullo studio dell'adeguatezza di questa tecnologia e sull'opportunità di ibridarla con metodi simbolici al fine di migliorarne l'efficacia.

Parlando di reti neurali profonde, affamate di dati per loro natura, diventa centrale nella trattazione anche lo studio dei dataset con cui possono essere addestrate e che devono promuovere le caratteristiche precedentemente descritte. L'interesse principale, in questo ambito, è la predisposizione del dataset a facilitare la verifica del ragionamento svolto dalla rete.

2. Stato dell'arte

Le tecnologie qui riportate seguono le specifiche precedentemente fornite nella sezione di definizione del problema.

Nella sezione **Architetture notevoli** si descriveranno i sistemi che più recentemente hanno affrontato il problema in questione presentando soluzioni innovative, mentre nella sezione **Dataset notevoli** si parlerà delle librerie di esempi che più si avvicinano alle caratteristiche richieste. Al termine di ciascuna sezione, si troverà un piccolo confronto delle tecnologie precedentemente descritte sotto forma di riassunto schematico. Successivamente, in fase di analisi del problema, si tratteranno le medesime questioni in maniera più approfondita.

2.1. Architetture notevoli

Le architetture riportate, una per sezione, sono selezionate tra quelle più nuove che meglio risolvono il problema o che lo affrontano in maniera particolarmente notevole. Si rimanda alla sezione finale e, successivamente, all'analisi del problema per un confronto tra i diversi approcci.

2.1.1. ProofWriter

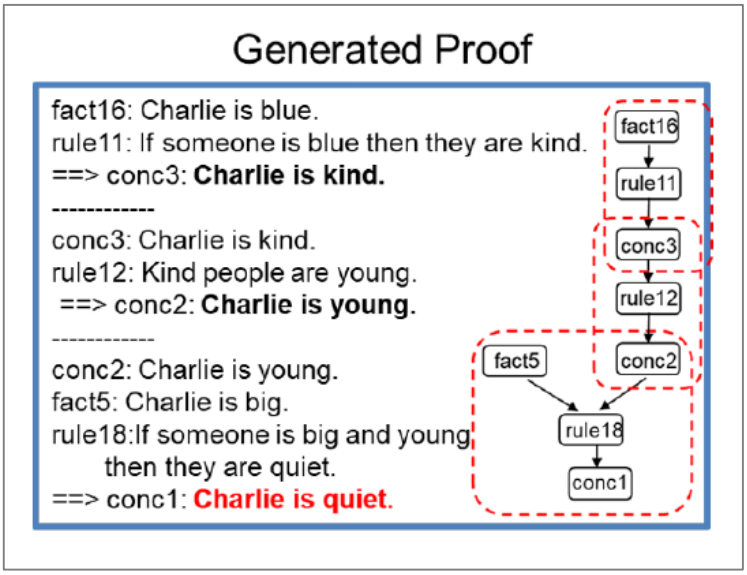
ProofWriter [17] è un modello neurale basato su una rete *T5* [12] e costituisce, probabilmente, il primo grande successo nel dimostrare la capacità delle reti *Transformer* [7] nell'ambito del ragionamento in linguaggio naturale dopo il parziale successo dei sistemi *RuleTaker* [18] e *PRover* [19], a cui esso si ispira.

Dato un elenco di frasi su cui ragionare, ovvero una teoria, e una domanda vero-falso sui concetti espressi, *ProofWriter* è in grado di generare la risposta e la dimostrazione formale del ragionamento compiuto. Sono supportati sia il ragionamento in CWA (Closed-World Assumption) con negation-as-failure che in OWA (Open-World Assumption), quindi contemplando anche l'eventualità che le risposte possano essere sconosciute oltre che vere o false.

La versione base di *ProofWriter* consiste semplicemente nel fornire teoria e domanda in ingresso alla rete neurale e ricevere in uscita risposta e dimostrazione, comprensiva degli eventuali passaggi intermedi. Più interessante è la versione iterativa, che genera una conclusione alla volta e riceve iterativamente in input la teoria e le proprie conclusioni precedenti fino ad esaurimento, poi ricostruisce la risposta finale in un secondo momento. Questa seconda versione generalizza meglio su teorie e dimostrazioni più lunghe rispetto a quelle viste in fase di addestramento [17].

La versione iterativa funziona leggermente meglio anche quando esposta a un linguaggio più sofisticato rispetto a quello di addestramento [17]. Questo modello, tra l'altro, fornisce sempre la dimostrazione che ha usato per produrre la risposta, mentre il sistema base potrebbe fornirne una non corrispondente, a meno di fargli dimostrare separatamente ciascun passaggio in un secondo momento.

Il modello iterativo risulta però piuttosto inefficiente in mancanza di un procedimento per guidare l'ordine di risoluzione, come comune tra i metodi forward-chaining, che costruiscono la dimostrazione avviando la ricerca a partire dalle frasi contenute nella teoria. Nei test effettuati dai ricercatori è inoltre capitato più volte che si superasse il limite di lunghezza per la sequenza in input imposto a default dal *Transformer*, con la conseguente necessità di troncare la teoria prima di passarla alla rete.



Funzionamento concettuale del ProofWriter. Figura tratta da [17].

Si riporta, infine, che nonostante il *ProofWriter* iterativo e i modelli simili siano in grado di fornire la dimostrazione effettivamente utilizzata per produrre la risposta, è stato evidenziato, per esempio nell'ambito dello studio che ha prodotto il linguaggio *MetaQNL* [20], come la correttezza di tali dimostrazioni non sia né garantita né sistematicamente verificabile.

2.1.2. NLProofS

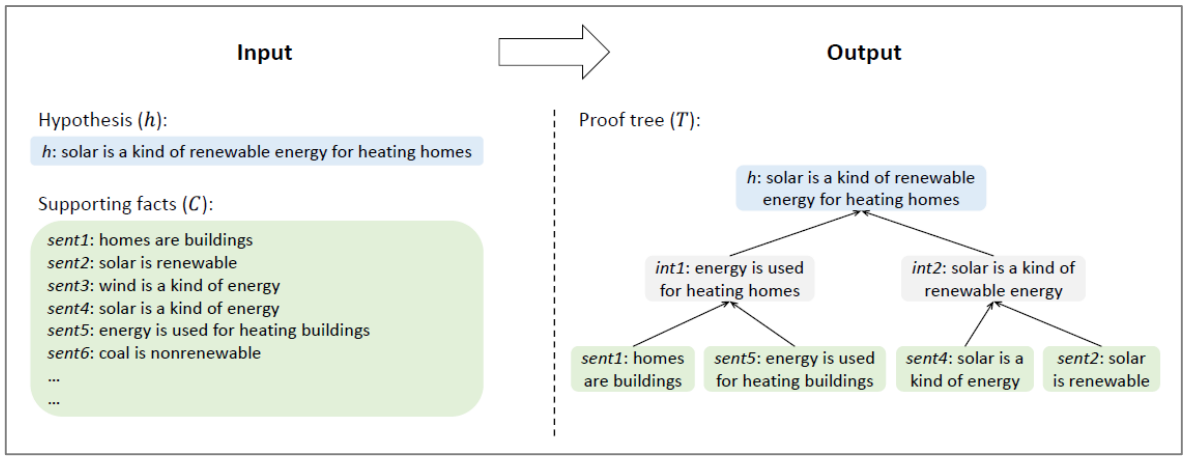
NLProofS (Natural Language Proof Search) [21] è formato da due componenti principali: il dimostratore, che genera uno alla volta i passaggi candidati ad essere inclusi nella dimostrazione, e il verificatore, che ne valuta la validità.

Il dimostratore è un modello *T5* [12] sottoposto a fine-tuning. Gli errori sintattici e le premesse inammissibili sono identificati tramite euristiche e filtrati in fase di generazione della sequenza. Vengono trattati come non validi anche i ragionamenti ammissibili, ma che concluderebbero un fatto già presente nella teoria. Per minimizzare la generazione di passaggi irrilevanti, il dimostratore riceve in input anche l'ipotesi che sta cercando di dimostrare, oltre alla teoria da cui si tenta di derivarla.

Il verificatore è un modello *RoBERTa* [10] indipendente dal dimostratore e sottoposto anch'esso a fine-tuning. Questo componente prende in input ciascuno step di dimostrazione, composto di varie premesse ed una conclusione che ne segue immediatamente, e produce un punteggio indicante la sua validità.

In fase di inferenza, dimostratore e verificatore sono coordinati dall'algoritmo di ricerca della soluzione, che cerca in maniera simbolica percorsi validi all'interno del grafo delle possibili dimostrazioni. L'obiettivo è trovare il cammino tra i fatti della teoria e l'ipotesi in questione che massimizzi il punteggio aggregato dei passaggi di cui si compone.

Il punteggio di ciascun passaggio è la media tra il punteggio assegnato dal verificatore e il punteggio fornito dal dimostratore, inteso come il valore di confidenza dichiarato dal modello durante la fase di generazione della sequenza tramite beam-search.



Descrizione schematica del funzionamento ai morsetti di NLProofS. Figura tratta da [21].

La possibilità di mitigare il problema dell'affidabilità delle dimostrazioni trattando il percorso tra premesse e conclusioni come un problema di ricerca nello spazio degli stati è stato precedentemente esplorato da altri studi, ad esempio *SCSearch* [22], mentre la principale innovazione introdotta da *NLProofS* è la presenza del verificatore. È stato infatti mostrato, ad esempio dagli studi che hanno prodotto *FaiRR* [20] e *SCSearch*, che quando i dimostratori neurali hanno accesso all'ipotesi da dimostrare, essi tendono a proporre dimostrazioni errate nel tentativo di raggiungere la conclusione attesa a tutti i costi. L'idea del verificatore è proprio quella di contrastare questo comportamento pur permettendo al dimostratore di guidare la ricerca conoscendo l'ipotesi.

I ricercatori riportano [21], infine, che l'accuratezza di risposte e dimostrazioni fornite da *NLProofS* risulta in linea con quelle delle alternative simili e talvolta leggermente superiore. Il verificatore migliora i risultati del dimostratore in maniera piuttosto consistente, pur non essendo perfetto, mentre il componente che si dimostra più bisognoso di miglioramento resta il dimostratore.

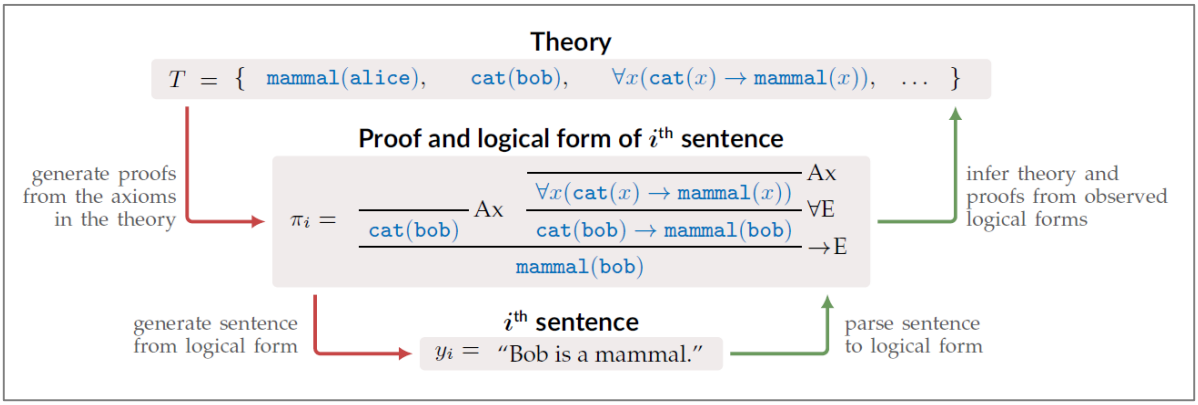
2.1.3. PWM e PWL

Il *Probabilistic Worldbuilding Model* (*PWM*) [5] è un modello bayesiano completamente simbolico che esprime tutti i passaggi, incluse le dimostrazioni, in un linguaggio formale direttamente interpretabile da un osservatore umano. L'approccio bayesiano è pensato per incentivare la capacità

di generalizzazione verso nuovi domini e nuovi task e la scelta di non ricorrere a tecniche sub-simboliche rende il sistema totalmente interpretabile. La combinazione di tecniche statistico-bayesiane di machine-learning con una rappresentazione simbolica permette di gestire l'incertezza in modo più accomodante rispetto ad un approccio simbolico tradizionale di tipo deterministico.

Mentre il *PWM* è una descrizione matematica dei componenti del sistema, il *Probabilistic Worldbuilding from Language* (*PWL*) [5] è l'algoritmo che riceve frasi in linguaggio naturale, calcola una rappresentazione logica del loro significato e aggiorna la teoria di conseguenza. Il suo componente di reasoning, dato un insieme di osservazioni da verificare, impiega un particolare tipo di abduzione [5] per cercare un set di assiomi in grado di spiegare deduttivamente tali osservazioni. L'uso di questo tipo di abduzione è spesso computazionalmente più leggero di una pura deduzione. La ricerca della dimostrazione avviene dunque in backward-chaining.

Durante l'inferenza, il *PWL* osserva una collezione di frasi con l'obiettivo di determinare il valore delle relative variabili. Il language-module determina la forma logica delle frasi e fornisce i valori più probabili delle variabili così individuate. Il reasoning-module determina poi il sottoinsieme della teoria da cui seguirebbero logicamente le forme logiche osservate e le relative dimostrazioni.



Funzionamento schematico di PWM (in rosso) e PWL (in verde). Figura tratta da [5].

Alle teorie più brevi e semplici viene assegnata una probabilità maggiore rispetto a quelle più complesse e le teorie inconsistenti vengono ulteriormente scoraggiate. È scoraggiata anche la decisione di attribuire diversi nomi ad una stessa entità, ma non la possibilità per un certo nome di riferirsi ad entità distinte.

Il modulo linguistico richiede che le regole di produzione non-terminali, ovvero conoscenze a priori circa la lingua inglese, siano specificate manualmente, tuttavia le stesse regole possono essere sfruttate in svariate applicazioni differenti, anche senza alcun cambiamento. La grammatica così costruita viene poi addestrata su un piccolo dataset, anch'esso costruito ed etichettato manualmente, per tarare automaticamente i parametri da cui derivano le conoscenze del modello sulla distribuzione statistica delle produzioni.

Per ottenere un risultato soddisfacente, nella pratica, è necessario selezionare manualmente il valore di diversi parametri in base al dataset che si intende utilizzare. Occorre inoltre che ciascuna parola che comparirà in fase di inferenza sia presente almeno una volta all'interno del dataset di addestramento, fanno eccezione i nomi propri.

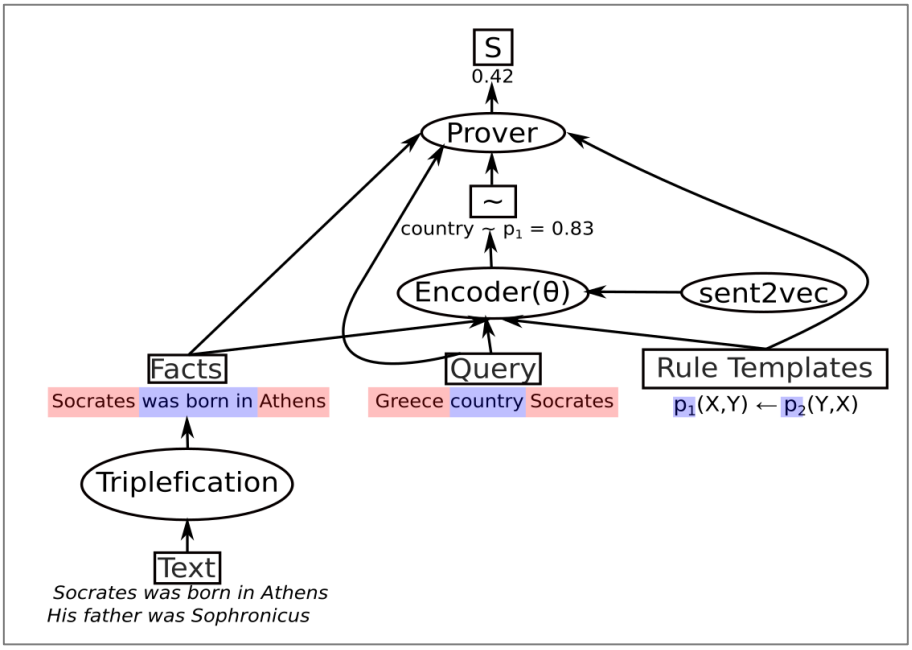
2.1.4. NLProlog

NLProlog [4] impiega un dimostratore di teoremi che sfrutta un meccanismo di backward-chaining analogo a quello impiegato nei sistemi Prolog tradizionali, dove però il confronto tra simboli è sostituito da una similarity-function differenziabile che produce un punteggio di unificazione al posto di una semplice risposta vero-falso.

Il testo in linguaggio naturale viene convertito in una rappresentazione formale in modo da poter essere passato al dimostratore logico. Le frasi vengono anzitutto trasformate in triple, dove il primo e il terzo elemento rappresentano le entità coinvolte, mentre il secondo elemento racchiude sotto forma di predicato il concetto espresso dalla frase che le lega. Tutti gli elementi nella tripla sono poi codificati in uno spazio vettoriale e le

rappresentazioni così ottenute vengono usate dalla similarity-function per calcolare le somiglianze tra entità e tra relazioni.

Le entità entro le frasi vengono riconosciute tramite il sistema statistico *spaCy* [24] e le triple vengono generate estraendo tutte le coppie di entità che appaiono in una stessa frase, usando il resto della frase come predicato.



Schema delle interazioni tra i componenti di NLProlog. Figura tratta da [4].

In linguaggio naturale, uno stesso concetto può essere espresso in diversi modi. Per risolvere questo problema si sostituisce il matching esatto tra simboli nell'operatore di unificazione Prolog con un operatore fuzzy che impiega la similarity-function per determinare quando unificare simboli differenti.

Il backward-chaining con unificazione debole produce un insieme di dimostrazioni, ciascuna associata ad un punteggio che ne misura il grado di verità. Il punteggio finale della dimostrazione è il massimo tra i punteggi di tutte le dimostrazioni trovate.

Per la rappresentazione degli embedding si usa un sentence-encoder formato da un componente statico pre-addestrato e da un componente con

parametri appresi durante l'addestramento. La natura completamente differenziabile del sistema consente di addestrare *NLProlog* tramite back-propagation sulla base dell'errore di predizione.

Le regole da utilizzare per trattare il linguaggio naturale vengono apprese da un dataset di addestramento. L'utente definisce un set di regole sotto forma di template con una certa struttura e *NLProlog* apprende dai dati gli embedding dei predicati così specificati. Le regole predefinite sono molto semplici e generiche, ma agendo su questo set è possibile incorporare domain-knowledge riguardante il task specifico.

2.1.5. FaiRR

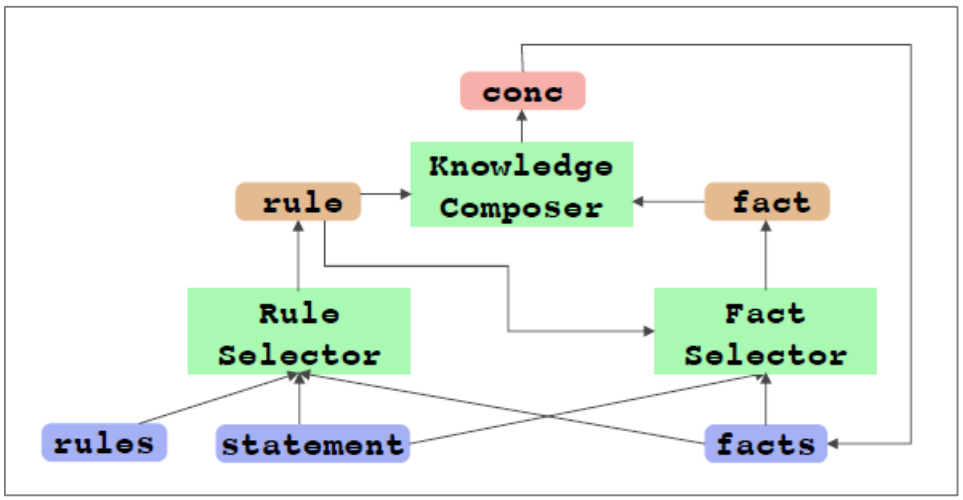
FaiRR (Faithful and Robust Reasoner) [23] è un modello iterativo formato da tre componenti modellati da *Transformer* [7] e addestrati indipendentemente. Un quarto componente deterministico cerca infine la risposta e ne ricostruisce la dimostrazione.

Il rule-selector è una rete *RoBERTa* [10] che riceve in input l'affermazione da dimostrare e l'intero elenco di fatti e regole della teoria, incluse le conclusioni intermedie già prodotte dal sistema, e seleziona una regola da utilizzare per lo step di inferenza successivo.

Il fact-selector, anch'esso un modello *RoBERTa*, prende in input l'affermazione, la regola selezionata e tutti i fatti della teoria, anche in questo caso includendo quelli prodotti dal sistema, e fornisce l'insieme dei fatti da usare unitamente alla regola per generare una nuova conclusione.

Il knowledge-composer è un modello *T5* [12] che riceve i fatti e la regola selezionati e produce una conclusione intermedia da aggiungere alla teoria sotto forma di nuovo fatto. Questo componente ragiona usando una sola regola e i soli fatti rilevanti. Ci si assicura in questo modo che il modello generi le proprie conclusioni a partire unicamente dai dati che saranno dichiarati nella dimostrazione e non da eventuali pattern nascosti nell'intera teoria. Ciascun singolo passo d'inferenza risulta così anche più semplice da apprendere e più robusto.

Il solver è il componente finale che opera al termine di tutte le iterazioni e cerca l'affermazione da dimostrare e il suo contrario tra tutte le inferenze generate. Il risultato può dunque essere confermato, confutato o sconosciuto. Il solver è un componente deterministico che non richiede addestramento.



Schema del procedimento di ragionamento in FaiRR. Figura tratta da [23].

FaiRR dimostra un'accuratezza paragonabile o leggermente inferiore rispetto alle soluzioni alternative più simili, produce però, in genere, risultati superiori nel caso di perturbazioni alla teoria non previste durante l'addestramento. *FaiRR* si dimostra anche molto efficiente in termini di velocità e di risorse computazionali, grazie alla lunghezza limitata delle sequenze che la rete *T5* deve gestire e al fatto che i due moduli di selezione conoscono a priori l'affermazione che il sistema sta cercando di dimostrare. Il problema dell'early-stopping, ovvero la tendenza di una rete a smettere di produrre nuove inferenze quando in realtà ve ne sarebbero ancora, è particolarmente accentuato in *FaiRR*, ma presente e significativo anche negli altri modelli.

2.1.6. Confronto delle architetture

Come primo confronto tra i sistemi, sono riportate una prima tabella che ne riassume le caratteristiche architetturali ed una seconda che compara le loro prestazioni secondo vari metri di giudizio. Prima di ciascuna tabella è descritto il significato delle chiavi di valutazione selezionate.

Tipo: "simbolico" indica che il funzionamento del sistema è totalmente interpretabile. "sub-simbolico" indica che il sistema fa principalmente uso di reti neurali o altre tecniche che rendono particolarmente complicato interpretare le motivazioni che hanno spinto il sistema a fornire un certo risultato. "ibrido" indica che il sistema combina tecniche simboliche e sub-simboliche.

Direzione: "forward" indica che il sistema costruisce la dimostrazione a partire dalle frasi della teoria, cercando di raggiungere logicamente l'ipotesi da dimostrare. "backward" indica che il sistema parte dall'ipotesi e tenta di determinare a ritroso quali segmenti della teoria sono necessari alla dimostrazione.

Architettura: Brevissima descrizione dei componenti principali e delle rispettive funzioni. Spiegazione dettagliata nei capitoli precedenti.

Modello	Tipo	Direzione	Architettura
ProofWriter	sub-simbolico / ibrido	forward	T5 riceve iterativamente in input le proprie conclusioni precedenti + ricerca della risposta alla fine
NLProofS	ibrido	forward	dimostratore T5 + verificatore RoBERTa + algoritmo di ricerca nel grafo costruito
PWM + PWL	simbolico	backward	abduce un sottoinsieme della teoria in grado di spiegare deduttivamente l'ipotesi
NLProlog	ibrido	backward	conversione frasi in entità e predicati + risolutore Prolog con unificazione tra simboli diversi, punteggio anziché V/F
FaiRR	ibrido	forward	rule-selector RoBERTa + fact-selector RoBERTa + knowledge-composer T5 + solver deterministico

Semplicità: Esprime quanto è facile comprendere e alterare il funzionamento dell'architettura in riferimento ai suoi componenti simbolici. I componenti neurali sono intesi come scatole nere e ignorati.

Interpretabilità: Esprime con quanta chiarezza è possibile comprendere in che modo il sistema produce le proprie conclusioni. È presa in considerazione anche la presenza di componenti neurali, che risultano tipicamente difficili da interpretare.

Affidabilità: Esprime qualitativamente il grado di sicurezza che ci si può aspettare da risposte e dimostrazioni fornite dal sistema.

Autonomia: Esprime quanto è probabile che l'architettura si adatti automaticamente a forme di linguaggio non esplicitamente contemplate durante lo sviluppo del sistema.

Precisione + Efficienza: Confrontare le prestazioni di queste architetture è piuttosto complicato poiché non sono disponibili esperimenti comparabili ed esaustivi. Spiegazione dettagliata nel capitolo successivo di analisi del problema.

Modello	Semplicità	Interpretabilità	Affidabilità	Autonomia	Precisione + Efficienza
ProofWriter	★★★	★☆☆	★☆☆	★★★	?
NLProofS	★★☆	★☆☆	★★☆	★★★	?
PWM + PWL	★☆☆	★★★	★★★	★☆☆	?
NLProlog	★★☆	★★☆	★★☆	★★☆	?
FaiRR	★★☆	★★☆	★★☆	★★★	?

2.2. Dataset notevoli

I dataset qui riportati, uno per sezione, affrontano tutti, con diversi gradi di precisione, il problema del ragionamento deduttivo in linguaggio naturale con dimostrazione. Si rimanda alla sezione finale e, successivamente, all'analisi del problema per un confronto tra i diversi approcci.

2.2.1. RuleTaker – ProofWriter

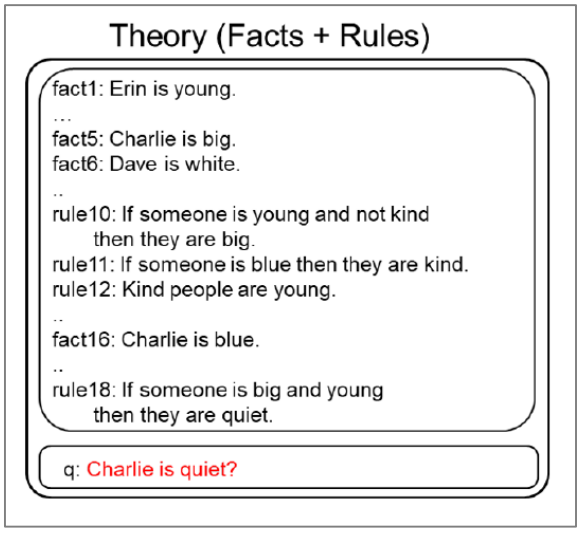
Il dataset *RuleTaker* [18], anche noto come dataset *ProofWriter* [17] nella sua versione riveduta e ampliata in occasione dell'omonimo studio, si compone principalmente di sei blocchi chiamati D0, D1, D2, D3, D4 e D5, ciascuno contenente circa 100˙000 domande in linguaggio naturale, con relativa teoria, generate proceduralmente. Il numero nel nome indica la profondità massima di ragionamento richiesta dal relativo split, ovvero il numero di applicazioni di regole richiesti per passare dalle frasi nella teoria alla proposizione cercata.

Vi sono inoltre il blocco Birds-Electricity, composto da circa 150 esempi scritti a mano e il blocco ParaRules, circa 40˙000 domande su 2˙000 teorie generate proceduralmente e poi parafrasate a mano in forma libera.

La dimostrazione è fornita come passaggi di deduzione in un grafo diretto aciclico che alterna fatti a regole. Le conclusioni intermedie sono presenti.

Le domande sono tutte del tipo vero-falso, sia in CWA che in OWA, e prevedono anche la gestione della negazione. Opzionalmente, è disponibile una versione per esperimenti di abduzione.

Ciascuna teoria è formata da fatti e regole a cui sono associati coppie domanda-risposta e sono disponibili le dimostrazioni alternative quando vi è più di un modo di giungere alla stessa conclusione.



Un campione del dataset RuleTaker. Figura tratta da [17].

Il dataset è piuttosto grande, le dimostrazioni sono espresse in modo rigoroso e vengono supportati diversi tipi di ragionamento, tuttavia la sua generazione procedurale comporta pattern abbastanza ripetitivi.

Un altro fatto da notare è la presenza di molti fatti e regole formalmente corretti, ma che esprimono concetti estremamente distanti dalla realtà e dal senso comune. Ciò può essere visto come un vantaggio, se si desidera concentrarsi sul ragionamento esplicito escludendo del tutto quello implicito, ma anche come un difetto, se si vuole trarre il massimo dalle informazioni apprese nella fase di pre-addestramento di un modello neurale.

2.2.2. EntailmentBank

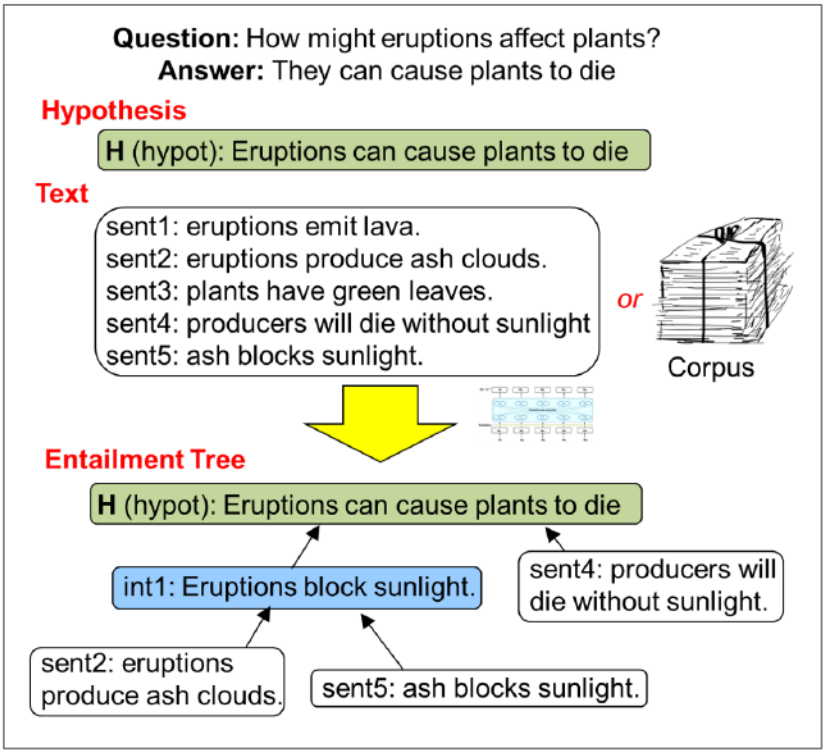
EntailmentBank [25] è un dataset molto piccolo, ma di elevata qualità, composto da circa 1˙840 domande con risposte in forma aperta, derivanti dal dataset *ARC* (AI2 Reasoning Challenge) [26], costituito da domande di scienza per studenti delle scuole elementari e medie americane.

La conoscenza è totalmente esplicita e in aggiunta a ciascuna coppia domanda-risposta è disponibile un'affermazione che riassume lo stesso concetto sotto forma di ipotesi, permettendo così di trattare il dataset anche come un insieme di soli esempi positivi a domande vero-falso in forma libera. Per creare esempi negativi è possibile perturbare alcuni esempi positivi rimuovendo premesse necessarie o sostituendole con distrattori, come proposto dai ricercatori di *NLProofS* [21]. Si noti che questo procedimento produce dimostrazioni invalide come esempi negativi, non dimostrazioni valide per ipotesi negate.

Le ipotesi, per costruzione, includono molto raramente frasi negate. La negazione compare però piuttosto spesso all'interno della teoria.

Le dimostrazioni sono redatte a mano da annotatori esperti e sono fornite sotto forma di alberi che descrivono le implicazioni a partire dai fatti della teoria, attraverso conclusioni intermedie, fino all'ipotesi espressa dalla domanda. Nella teoria non si fa distinzione tra fatti e regole e le dimostrazioni espongono i propri passaggi in modo preciso.

Sono opzionalmente disponibili una versione con distrattori nella teoria ed una versione la cui teoria non è composta da affermazioni separate, ma da un unico corpo di testo a tema scienza e cultura generale proveniente dal dataset *WorldTree v2* [27].



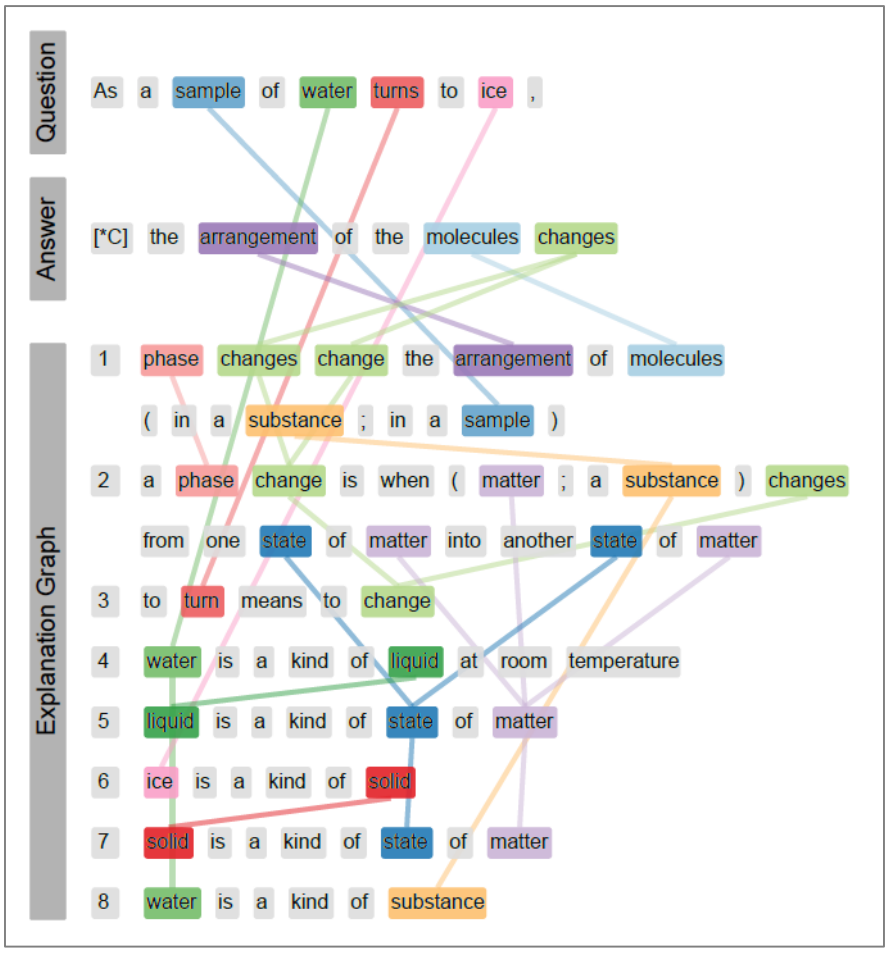
Esempio del tipico caso d'uso previsto da EntailmentBank. Figura tratta da [25].

2.2.3. WorldTree v2

WorldTree v2 [27] è un'espansione del progetto *WorldTree* [28] che propone circa 5˙000 domande a risposta multipla tratte dal dataset *ARC* [26].

Oltre alle domande è fornita un'ampia base di conoscenza contenente informazioni sia riguardo ai vari campi della scienza che riguardo a concetti più generici e di buonsenso, ma sempre in forma esplicita. Si tratta di circa 9˙000 fatti suddivisi per categoria in 66 tabelle.

Le dimostrazioni sono costruite a mano da esperti e fornite sotto forma di dettagliati grafi di spiegazione che citano le informazioni utili presenti nella base di conoscenza e ne evidenziano i termini in comune. Queste dimostrazioni sono informali, ma molto precise, e consentono di ragionare in maniera esplicita anche su questioni di buonsenso grazie all'ampia fornitura della base di conoscenza.



Un tipico grafo WorldTree che lega la domanda alla base di conoscenza. Figura tratta da [27].

La presenza di un'unica enorme e articolata base di conoscenza per tutte le domande, tuttavia, rende piuttosto complicata la gestione del dataset, che per certi versi risulta più adatto a problemi di ricerca e reperimento delle informazioni che non a problemi di ragionamento deduttivo.

2.2.4. e-SNLI

Il dataset *SNLI* (Stanford Natural Language Inference) [29] propone circa 570˙000 esempi composti a partire da didascalie scritte per immagini di carattere generale. *e-SNLI* [30] è un'estensione di *SNLI* che giustifica ciascuna risposta corretta con una spiegazione scritta da annotatori umani.

Sono fornite una premessa e un'ipotesi in linguaggio naturale e bisogna determinare se l'ipotesi segue logicamente dalla premessa (entailment), se la contraddice (contradiction) o se risultano scorrelate (neutral). La capacità di determinare questa etichetta richiede nozioni di buonsenso implicite.

Il dataset non fornisce dimostrazioni formali, ma solo spiegazioni in linguaggio naturale e in forma libera che descrivono soggettivamente il collegamento tra le premesse e le ipotesi. Vi è una sola spiegazione per domanda nello split di addestramento, mentre sono fornite tre possibili spiegazioni valide per gli esempi negli split di validazione e test.

Dato che i contenuti di questo dataset provengono da un gran numero di annotatori inesperti differenti, è possibile trovare, non troppo di rado, errori grammaticali e spiegazioni incomplete o discutibili. Tuttavia, l'enorme dimensione del dataset e la grande varietà di stili dovrebbero essere sufficienti ad addestrare modelli con buone prestazioni.

2.2.5. CoS-E

CoS-E (Common Sense Explanations) [31] è un dataset formato aggiungendo annotazioni umane ai segmenti di addestramento e di sviluppo del dataset *CommonsenseQA* [32] in versione random-split.

Il dataset si compone di circa 11˙000 esempi formati da un identificatore per rintracciare la domanda in *CommonsenseQA* (con 3 o 5 possibili scelte per la soluzione), una spiegazione aperta della risposta corretta e l'annotazione della porzione di testo ritenuta cruciale per la spiegazione. La capacità di rispondere alle domande richiede buonsenso implicito.

Gli esempi nel dataset non dispongono di una vera e propria dimostrazione, ma solo di una spiegazione informale che giustifica la scelta della risposta corretta tra quelle date. Tali spiegazioni sono prodotte da annotatori umani non specializzati e in molti casi risultano confusionarie o fuorvianti. Tuttavia, secondo lo studio, il dataset consente comunque l'addestramento di modelli funzionali. Ciò è dovuto, probabilmente, al numero elevato di esempi, se si considera l'origine non procedurale dei dati, e alla varietà di stili derivante dalla moltitudine di autori.

2.2.6. Confronto dei dataset

Per un primo confronto tra i dataset, sono riportate due tabelle che ne riassumono le caratteristiche principali evidenziando le differenze. Prima

di ciascuna tabella è descritto il significato delle chiavi di valutazione selezionate.

Vocabolario: "generico" indica che il dataset non tratta un tema ben specifico né argomenti particolarmente esperti. "scienza" indica che il dataset esprime quasi esclusivamente concetti in ambito scientifico.

Costruzione: "procedurale" indica che il dataset è composto da frasi prodotte automaticamente da un algoritmo in base a dei template predefiniti. "autori esperti" indica che gli esempi sono stati scritti con attenzione da ricercatori con una profonda conoscenza delle caratteristiche richieste. "autori amatoriali" indica che i campioni sono stati prodotti in massa da autori inesperti e potenzialmente ignari del caso d'uso, tipicamente raggiunti tramite crowdsourcing.

Dimostrazione: "rigorosa" indica una dimostrazione scomposta in singoli passaggi piuttosto specifici. "informale" indica una spiegazione del ragionamento troppo generica per poter essere definita dimostrazione. "ibrida" indica una dimostrazione comunque informale, ma piuttosto precisa.

Formato: Brevissima descrizione del formato dei dati. Spiegazione dettagliata nei capitoli precedenti.

Dataset	Vocabolario	Costruzione	Dimostrazione	Formato
RuleTaker	generico	procedurale	rigorosa	teoria (fatti e regole) + domande V/F + dimostrazioni alternative
EntailmentBank	scienza	autori esperti	rigorosa	teoria + domande a risposta aperta (o ipotesi) + dimostrazione
WorldTree	scienza	autori esperti	ibrida	domande a risposta multipla + base di conoscenza
e-SNLI	generico	autori amatoriali	informale	premessa + ipotesi + etichetta (entail, contradict, neutral) + spiegazione/i
Cos-E	generico	autori amatoriali	informale	ID in CQA (domande a risposta multipla) + spiegazione

Conoscenza: "esplicita" indica che tutte le informazioni necessarie a compiere i ragionamenti richiesti sono esplicitamente dichiarate nella teoria. "implicita" indica che per completare i ragionamenti è richiesta una certa quantità di buonsenso e di conoscenza del mondo.

Completezza: "open world" indica che la risposta a una domanda può essere sconosciuta, ovvero che un'ipotesi può essere né confermata né confutata dalla teoria. "negazione parziale" indica che le frasi possono contenere la negazione, tuttavia le ipotesi sono sempre dimostrabili positivamente.

Opzionali: Brevissimo elenco di caratteristiche aggiuntive, spesso assenti dal dataset principale, ma presenti come possibili alternative. Spiegazione dettagliata nei capitoli precedenti.

Esempi: Numero indicativo di campioni distinti presenti nel dataset.

Homepage: Link al sito ufficiale da cui può essere scaricato il dataset.

Dataset	Conoscenza	Completezza	Opzionali	Esempi	Homepage
RuleTaker	esplicita	open world	CWA + abduzione + teoria a mano + corpus	100˙000 (+40˙000)	allenai.org/data/proofwriter
EntailmentBank	esplicita	negazione parziale	distrattori + corpus	1˙800	allenai.org/data/entailmentbank
WorldTree	esplicita	negazione parziale	(common sense esplicito)	5˙000	cognitiveai.org/explanationbank
e-SNLI	implicita	open world	parole chiave	570˙000	github.com/OanaMariaCamburu/e-SNLI
Cos-E	implicita	negazione parziale	parole chiave	11˙000	github.com/salesforce/cos-e

3. Analisi del problema

Questo capitolo presenta un'analisi più approfondita della situazione complessiva introdotta nei capitoli precedenti. L'obiettivo principale è quello di esplicitare tutte le complicazioni incontrate e le principali strade che possono condurre ad un miglioramento delle architetture presentate.

Nella sezione **Le criticità dei linguaggi naturali** si discuteranno i principali problemi che è necessario affrontare lavorando con i dataset precedentemente illustrati e si presenteranno alcune delle soluzioni proposte dai relativi articoli. Nella sezione **Il confronto delle architetture stato dell'arte** si spiegherà perché è così difficile comparare tali architetture e si proporranno alcuni parziali confronti quantitativi laddove è possibile. Infine, nella sezione **Ricostruzione dell'architettura ProofWriter** si proporrà una motivata ricreazione delle porzioni più significative dell'omonimo studio.

3.1. Le criticità dei linguaggi naturali

È difficile pensare che un software possa essere in grado di compiere qualsiasi altra forma di ragionamento logico con risultati soddisfacenti senza prima aver conquistato ottimi risultati nel ragionamento deduttivo. Gran parte dei dataset composti per studiare il ragionamento si concentrano infatti su quello deduttivo ed offrono solo un supporto parziale a forme di ragionamento più incerte.

Al fine di sviluppare un sistema affidabile, non è sufficiente assicurarsi che le risposte dedotte siano corrette, ma è necessario poter verificare che anche i procedimenti che le producono siano logicamente validi. I dataset che permettono di verificare la bontà dei ragionamenti sono tuttavia molto pochi. Il problema risiede nel fatto che la raccolta e catalogazione di dimostrazioni oggettive è un compito particolarmente costoso. Il risultato è che più un dataset è grande, più la sua qualità è limitata, come nel classico dilemma della coperta corta. All'aumentare del numero di esempi, infatti, le dimostrazioni non possono più essere curate da un team di esperti, così

è necessario affidarsi a un gran numero di annotatori inesperti o, eventualmente, a metodi di generazione sub-simbolici, rinunciando così alla rigorosità delle dimostrazioni e accettando una certa percentuale di errori nel dataset. Un'alternativa differente, ma altrettanto problematica, consiste nell'affidarsi alla generazione procedurale, rinunciando ad un certo grado di varietà nel lessico e nella costruzione delle frasi, ma assicurando un grado di correttezza e precisione impeccabile.

Il dataset *RuleTaker* [18], ad esempio, propone un approccio a generazione procedurale con dimostrazione formale e un numero elevato di campioni. Di seguito, a titolo di esempio, un campione rappresentativo, riformattato per favorirne la leggibilità:

```
Triples:
triple1: Charlie is red.
triple2: Dave is furry.
triple3: Dave is kind.
triple4: Dave is not nice.
triple5: Gary is furry.
triple6: Gary is kind.
triple7: Gary is rough.
Rules:
rule1: If Gary is rough then Gary is furry.
rule2: All red, rough things are furry.
rule3: If Gary is green and Gary is kind then Gary is rough.
rule4: Nice, red things are cold.
rule5: Red things are nice.
rule6: All furry, kind things are not nice.
rule7: If something is red and nice then it is rough.
rule8: If something is green and nice then it is rough.
Question:
Gary is not nice.
Answer:
true
Proofs:
(triple5 triple6) -> (rule6 % int1)
(((triple7) -> (rule1 % int2)) triple6) -> (rule6 % int1)
int1: Gary is not nice.
int2: Gary is furry.
```

EntailmentBank [25] sceglie invece di avere un numero molto limitato di esempi, ma curati da annotatori esperti e con dimostrazione formale:

```
Triples:
sent1: a leaf performs photosynthesis
sent2: carbohydrates are made of sugars
sent3: photosynthesis makes food for the plant by converting carbon
dioxide, water, and sunlight into carbohydrates
sent4: sunlight is a kind of solar energy
Question:
Which of these is a function of a leaf?
Answer:
Converting solar energy into sugar
Hypothesys:
a leaf converts solar energy into sugar
Proof:
sent2 & sent3 -> int1: photosynthesis converts sunlight into sugar
int1 & sent4 -> int2: photosynthesis converts solar energy into sugar
int2 & sent1 -> hypothesis
```

Una scelta completamente opposta è rappresentata, per esempio, da *e-SNLI* [30], che fornisce solo spiegazioni informali e soggettive, ma consente di avere a disposizione un gran numero di esempi:

```
Premise:
a white tented fruit stand with several people shopping in it.
Conclusion:
A group of people are dancing in a circle.
Gold Label:
contradiction
Explanation:
People cannot be shopping and dancing simultaneously.
```

Lavorando in linguaggio naturale, è in ogni caso piuttosto difficile verificare la correttezza delle dimostrazioni e delle risposte a domande aperte, anche in presenza di un buon dataset. Questo poiché vi sono in genere tantissimi modi diversi di esprimere uno stesso concetto ed è poco realistico pensare che un dataset possa includere tutte le alternative.

È dunque necessario prevedere un sistema ausiliario in grado di stabilire se due frasi abbiano o meno lo stesso significato. Lo sviluppo di un simile componente rappresenta, volendo, una sfida a sé stante ancora

aperta. Esso andrebbe inevitabilmente ad introdurre nuova incertezza nel sistema poiché, in primo luogo, tale comparatore non sarebbe perfetto. In secondo luogo, anche in presenza di un comparatore ideale, l'intrinseca ambiguità delle rappresentazioni in linguaggio naturale renderebbe impossibile la totale eliminazione degli errori.

EntailmentBank, ad esempio, propone di impiegare *BLEURT* (Bilingual Evaluation Understudy with Representations from Transformers) [33], un modello neurale basato su *BERT* [8], per confrontare le frasi corrette con quelle generate dal sistema e decidere se esprimano o meno il medesimo significato. Nel caso del dataset *RuleTaker*, invece, vista la semplicità e regolarità delle frasi, *ProofWriter* [17] opta per un più semplice matching esatto tra stringhe.

Un altro importante distinguo da fare riguarda la quantità di buonsenso implicito, o common-sense, che il dataset richiede di gestire. Se è richiesto, ad esempio, che il sistema comprenda autonomamente che "*Il cane sta correndo*" e "*Il cane sta dormendo*" sono concetti in contraddizione parliamo di buonsenso richiesto, altrimenti parliamo di buonsenso non richiesto.

In molti casi, tuttavia, la distinzione non è altrettanto netta. Se ad esempio si forniscono al sistema informazioni su una persona chiamata *Franco* e ci si riferisce successivamente a lui come *Frank*, è lecito per il sistema assumere che si tratti di soggetti diversi? Un sistema senza buonsenso deve comunque essere in grado di riconoscere i sinonimi? Un sistema senza buonsenso deve essere in grado di identificare le coniugazioni dei verbi irregolari? Dove tracciare la linea tra buonsenso e semplice funzionamento e, soprattutto, come assicurarsi che un modello sub-simbolico non oltrepassi tale linea resta un problema aperto.

Un'ultima questione da sottolineare riguarda l'individuazione del contrario di una certa affermazione in modo da poter affermare la falsità di un'ipotesi. Al di là della generica capacità di individuare il contrario di un concetto, che si riconduce al problema del buonsenso appena definito, è comunque necessario domandarsi come individuare le forme più semplici

di negazione. Per un dataset schematico come il *RuleTaker* può essere ad esempio sufficiente aggiungere e rimuovere le negazioni esplicite tramite espressioni regolari, come proposto dai ricercatori di *ProofWriter*. Si potrebbe alternativamente ricorrere ad un modello neurale dedicato a questo compito, come già discusso per il problema opposto di riconoscimento di frasi distinte con lo stesso significato.

Riassuntivamente, si individuano dunque i seguenti problemi:

Problema	**Compromessi proposti**	**Esempi**
poco supporto al ragionamento non deduttivo	(il focus sul ragionamento deduttivo è complessivamente ragionevole)	-
più un dataset è grande e più la qualità scarseggia	accontentarsi di un dataset di piccole dimensioni	*EntailmentBank*
	rinunciare alla rigorosità delle dimostrazioni e accettare la presenza di errori	*e-SNLI*
	rinunciare alla varietà nel linguaggio	*RuleTaker*
pochi dataset consentono di verificare la correttezza del ragionamento	(si riconduce al problema precedente)	-
uno stesso concetto può essere espresso in molti modi differenti	impiegare un componente sub-simbolico per confrontare le frasi e accettare un certo grado di ambiguità	*EntailmentBank*
	affidarsi al matching esatto tra stringhe e rinunciare alla varietà di linguaggio	*RuleTaker*
determinare il grado di buonsenso implicito che bisogna gestire	(problema ancora molto aperto)	-
è difficile individuare il contrario di un'affermazione (in generale si riconduce al problema precedente)	individuare le forme semplici di negazione tramite espressioni regolari e rinunciare alla varietà nel linguaggio	*RuleTaker*
	impiegare un componente sub-simbolico e accettare la presenza di errori	-

3.2. Il confronto delle architetture stato dell'arte

Un confronto equo e definitivo al fine di classificare le prestazioni delle varie architetture deve essere anzitutto svolto su uno stesso dataset. All'interno dello stesso dataset è inoltre necessario assicurarsi di utilizzare lo stesso setup, dato che vi sono molti modi differenti di utilizzare un dataset complesso. Bisogna poi accordarsi sull'uso delle stesse metriche di confronto e, nel caso delle architetture che richiedono una certa quantità di conoscenza sul dominio, è necessario stabilire quanto questa conoscenza influisca sul risultato. Un sistema che richiede la definizione manuale del dizionario da utilizzare, per esempio, potrebbe produrre un ottimo risultato sul dataset impiegato per i test, ma un risultato significativamente peggiore in un'applicazione reale. In questo contesto, un sistema meno performante, ma con parametri più generali e automaticamente appresi potrebbe produrre risultati più soddisfacenti.

Il benchmark più comunemente utilizzato nell'ambito del ragionamento in linguaggio naturale con dimostrazione è al momento il dataset *RuleTaker* [18]. Esso è fornito, nella sua distribuzione più recente, in più di 30 versioni distinte, differenziate in base a parametri come il tipo di ragionamento richiesto, il numero massimo di passaggi intermedi e la ricchezza della sintassi. Nulla vieta, inoltre, che si utilizzi una versione per l'addestramento e una differente per il test di una stessa architettura. Questo capita, ad esempio, se si decide di addestrare il modello per una profondità di ragionamento ridotta e testarlo su una più elevata o se si vuole addestrarlo a compiere uno step alla volta e poi testarlo iterativamente su dimostrazioni a più passaggi.

RuleTaker, tra l'altro, fa una netta distinzione all'interno delle proprie teorie tra fatti e regole. È quindi possibile scegliere se nascondere o meno questa distinzione al modello, così come è possibile fornire le frasi nell'ordine di default o permutarle casualmente a ciascuna iterazione. Inoltre, nel caso delle teorie che supportano più domande, si può pensare di permutare differentemente le frasi per ciascuna domanda.

Sebbene un confronto esaustivo tra queste architetture sia al momento assente, e, come descritto, estremamente complicato, è anche vero che alcuni confronti localizzati esistono e possono aiutare a farsi un'idea sullo stato dell'arte.

FaiRR [23], ad esempio, riporta i propri risultati con addestramento e test sugli split D0-D3 OWA del dataset *RuleTaker* in confronto ai risultati ottenuti da un modello *ProofWriter* iterativo [17]. *NLProofS* [21] ha successivamente aggiunto anche i propri risultati al benchmark:

Modello	Risposte corrette	Dimostrazioni corrette
ProofWriter	99,8 %	99,7 %
NLProofS	99,6 %	99,5 %
FaiRR	99,2 %	98,8 %

Si noti che il *ProofWriter* considerato nel benchmark è basato su una rete *T5* [12] molto più leggera, ma meno performante, rispetto a quella prevista dal *ProofWriter* base. Il motivo di questa scelta è l'equità di confronto con le altre due architetture che usano, appunto, quella stessa rete. Stando a questo specifico confronto, *ProofWriter* risulta leggermente superiore, tuttavia, considerata la vicinanza numerica tra i punteggi e dato uno sguardo ai dati disaggregati, si può concludere un pari merito con *NLProofS*. Vista la specificità del confronto, si può anche più propriamente affermare che il risultato non è conclusivo.

FaiRR riporta anche le proprie prestazioni in confronto a *ProofWriter* su una versione di *RuleTaker* D3 appositamente alterata sostituendo prima tutti i nomi, poi tutti gli aggettivi e infine entrambe le categorie con nuovi termini assenti nel dataset base. È possibile in questo modo farsi un'idea della robustezza di questi modelli quando sono esposti ad un nuovo dominio. In tabella è riportata la media fra i tre casi e viene anche valutata la consistenza dei modelli nel fornire le stesse risposte e dimostrazioni a fronte di perturbazioni differenti:

Modello	Risposte corrette	Dimostrazioni corrette	Consistenza
ProofWriter	94,1 %	93,1 %	92,9 %
FaiRR	96,3 %	95,3 %	95,9 %

In questo caso, le performance di *FaiRR* risultano superiori rispetto a quelle di *ProofWriter*.

Per quanto riguarda *PWM* e *PWL* [5], i ricercatori forniscono i risultati di alcuni test sul dataset *RuleTaker*, tuttavia unicamente per il blocco Birds-Electricity e, in ogni caso, con un setup difficilmente confrontabile. *NLProlog* [4], invece, riporta solo i risultati di test ottenuti su dataset estranei alle altre architetture e che non permettono la verifica della dimostrazione ottenuta, ma solo della risposta.

Un confronto quantitativo e conclusivo tra i sistemi trattati richiederebbe, evidentemente, nuovi test appositamente preparati e una grande quantità di lavoro al di là degli scopi di questa tesi. In compenso, uno sguardo d'insieme alle architetture proposte e alle relative problematiche suggerisce di approfondire l'analisi di *ProofWriter*, *NLProofS* e *FaiRR* al fine di risolvere alcuni problemi e migliorarne le prestazioni, eventualmente sfruttando le intuizioni presenti negli altri sistemi studiati. Queste tre architetture condividono la rete *T5* come componente fondamentale e soffrono tutte del problema dell'early-stopping. Il modo più logico di procedere sembra dunque essere uno studio dettagliato di *ProofWriter*, dato che, a fronte di prestazioni, funzionamento e difetti molto simili a quelli dei concorrenti, presenta un'architettura nettamente più semplice.

3.3. Ricostruzione dell'architettura ProofWriter

Il codice per effettuare l'addestramento e l'inferenza dei modelli *ProofWriter* [17] non è pubblicamente disponibile, tuttavia gli autori forniscono alcuni dei modelli *T5* [12] già addestrati e la descrizione del funzionamento data dall'articolo è sufficientemente dettagliata da permettere la creazione di un sistema equivalente all'originale.

Per la ricostruzione dell'architettura, ho impiegato principalmente il framework di machine-learning *PyTorch* [34] e il framework *HuggingFace* [35], specializzato nell'uso dei *Transformer* [7], entrambi disponibili in linguaggio Python.

3.3.1. Dataset

Ho modellato i dataset *RuleTaker* D* [18] per l'addestramento iterativo, cioè quelli che insegnano alla rete a compiere un passo d'inferenza alla volta, come specializzazioni della classe astratta *Dataset* di *PyTorch* [34]. Alla creazione dell'oggetto, viene letto il file di testo specificato contenente gli esempi e ciascun campione è caricato in memoria come elemento di una lista.

Ho implementato come opzioni la possibilità di limitare il dataset caricacato ad un certo numero di esempi e la possibilità di avviare il caricamento a partire da un certo offset. In questo modo è possibile sia effettuare test più veloci su un pool ridotto che separare un unico dataset in split più piccoli. Ho anche implementato la possibilità di verificare che le sequenze in input non superino la lunghezza massima accettata dal modello. In questo modo è possibile determinare la lunghezza che deve essere supportata e grazie all'aiuto del framework *HuggingFace* [35] è possibile configurare il modello di conseguenza, superando così una delle limitazioni del *ProofWriter* [17] originale.

La funzione che restituisce un certo esempio, specificandone l'indice, recupera la stringa corrispondente dalla lista e ne estrae fatti e regole, componendoli in un'unica teoria. Il dataset *RuleTaker*, infatti, distingue esplicitamente i fatti dalle regole, mentre il modello *ProofWriter* no. L'ordine delle frasi viene permutato casualmente e l'etichetta originale di ciascuna frase viene sostituita con una nuova etichetta che maschera la distinzione tra fatti e regole. L'etichetta originale viene comunque conservata in modo da poter confrontare le dimostrazioni restituite dal modello con quelle corrette.

Vengono estratti dalla stringa anche tutti i nuovi fatti che possono essere inferiti dalla teoria insieme alle relative dimostrazioni, incluse quelle alternative quando presenti. Le frasi nelle dimostrazioni sono identificate dall'etichetta originale, che viene sostituita anche in questo caso con quella nuova. Se non vi è alcuna inferenza deducibile dalla teoria, tale indicazione viene riportata come unica conclusione possibile.

Tra tutte le possibili inferenze e dimostrazioni valide, viene selezionata casualmente una coppia da utilizzare come risposta corretta durante l'addestramento del modello. Quando il modello viene testato, è invece possibile confrontare la risposta calcolata con tutte le coppie valide.

Le sequenze di input per il modello e le risposte corrette vengono restituite dalla funzione già convertite nella rappresentazione numerica richiesta dal modello tramite l'implementazione specificata del *Tokenizer HuggingFace*. Vengono restituiti anche l'elenco di tutte le risposte valide e il testo originale sotto forma di stringa.

3.3.2. Addestramento

Per quanto riguarda il ciclo d'addestramento, ho optato per utilizzare le funzioni standard *PyTorch* [34] anziché la metodologia semplificata proposta da *HuggingFace* [35] in modo da avere maggior controllo sui dettagli dell'implementazione, nell'ottica di studiare il sistema ad un maggior livello di dettaglio e poter alterare qualsiasi aspetto del suo funzionamento.

Per ogni epoca di addestramento, il sistema viene allenato su ciascun campione nel relativo dataset, scegliendo ogni volta una possibile inferenza e una possibile sua dimostrazione e randomizzando l'ordine delle frasi nella teoria. L'addestramento avviene tramite back-propagation sulla base della funzione di loss entro un batch di dimensioni impostabili dall'utente. Anche l'ordine degli esempi, e quindi la composizione del batch, è casuale.

La verifica dei risultati di ciascuna epoca avviene misurando l'accuratezza delle previsioni sul dataset di validazione. Il risultato di ciascun

esempio nel test è considerato corretto solo se la coppia risposta-dimostrazione fornita risulta uguale, lettera per lettera, a una di quelle indicate come corrette. È infatti estremamente raro che il modello neurale fornisca la risposta corretta sbagliandone unicamente il formato.

Il supporto del framework *HuggingFace* consente la taratura di diversi parametri che influiscono sulla generazione del risultato e che sono stati lasciati a default nel *ProofWriter* [17] originale poiché non erano altrettanto facilmente impostabili senza *HuggingFace*. In particolare, il parametro *num_beams* indica quante sequenze distinte considerare nella ricerca di quella con il valore di confidenza più alto, mentre i parametri *length_penalty* e *repetition_penalty* regolano quanto debbano essere scoraggiate le sequenze molto lunghe o che presentano ripetizioni. Ho optato per sperimentare su un piccolo campione tutte le combinazioni dei valori più comunemente utilizzati e ho scelto quella che fornisce più consistentemente i risultati migliori:

```
num_beams = 4
repetition_penalty = 2.5
length_penalty = 0.5
```

Per la scelta dell'ottimizzatore, ho seguito il medesimo procedimento e sperimentato su un campione ridotto le combinazioni più popolari. È risultato vincente l'ottimizzatore *Adafactor* [36].

Il ciclo di addestramento consente di salvare l'ultima versione del modello addestrato e la versione che di volta in volta si dimostra migliore in fase di valutazione. Assieme ai pesi del modello viene salvato anche lo stato dell'ottimizzatore, in modo da poter riprendere e proseguire l'addestramento in un secondo momento. Alla fine di tutti i cicli, le due versioni del modello vengono messe alla prova sul dataset di test.

3.3.3. Inferenza

Il *ProofWriter* iterativo [17] è addestrato nel compito di aggiungere una nuova conclusione valida alla teoria data. In fase di inferenza, una teoria viene ripetutamente sottoposta al modello fino ad esaurimento di nuove

conclusioni e, in un secondo momento, si cercano tra le conclusioni generate l'affermazione richiesta e il suo contrario, al fine di dar risposta al quesito.

Per compiere la ricerca di un'affermazione e del suo contrario, lo studio originale propone una semplice soluzione basata su espressioni regolari che aggiunge e rimuove la parola *not* dalle frasi.

Io ho predisposto una soluzione con validità un po' più generale, pur sempre nell'ambito dei dataset sulla falsariga di *RuleTaker* [18], impiegando la libreria *spaCy* [24] per separare le frasi in token, scompattando anche le forme abbreviate e coniugate, come ad esempio *doesn't*, nei loro componenti fondamentali, in questo caso *do* e *not*. Se la frase contiene la parola *not*, ne creo la versione opposta rimuovendo tale particella. Se il token subito precedente a *not* è *do*, creo una seconda versione opposta rimuovendo anch'esso.

Invertire una frase affermativa richiederebbe, invece, di determinare in quale posizione inserire la negazione. Per ovviare a questo problema, ho preferito lasciare tali frasi in forma affermativa e rimuovere piuttosto la negazione dalle altre frasi con cui devono essere confrontate. Questa operazione risulta, infatti, equivalente, dal momento che in un sistema che non fa uso di buonsenso, la confutazione di una frase richiede necessariamente la presenza di una negazione esplicita in sé stessa o nel proprio opposto.

Si noti che questa gestione della negazione, e in particolare la rimozione della particella *not*, risulta efficace per il dataset *RuleTaker* poiché tale operazione è pensata per essere svolta sui fatti, i quali sono sempre riferiti a soggetti individuali. La quantificazione universale, ad esempio, è utilizzata solo nel caso delle regole, le quali non necessitano di essere negate e confrontate.

Così gestita la negazione, ho poi predisposto sia una funzione per eseguire l'inferenza completa con tutti i passaggi su un singolo esempio specificato manualmente che quella per eseguirla su un dataset apposito. Si

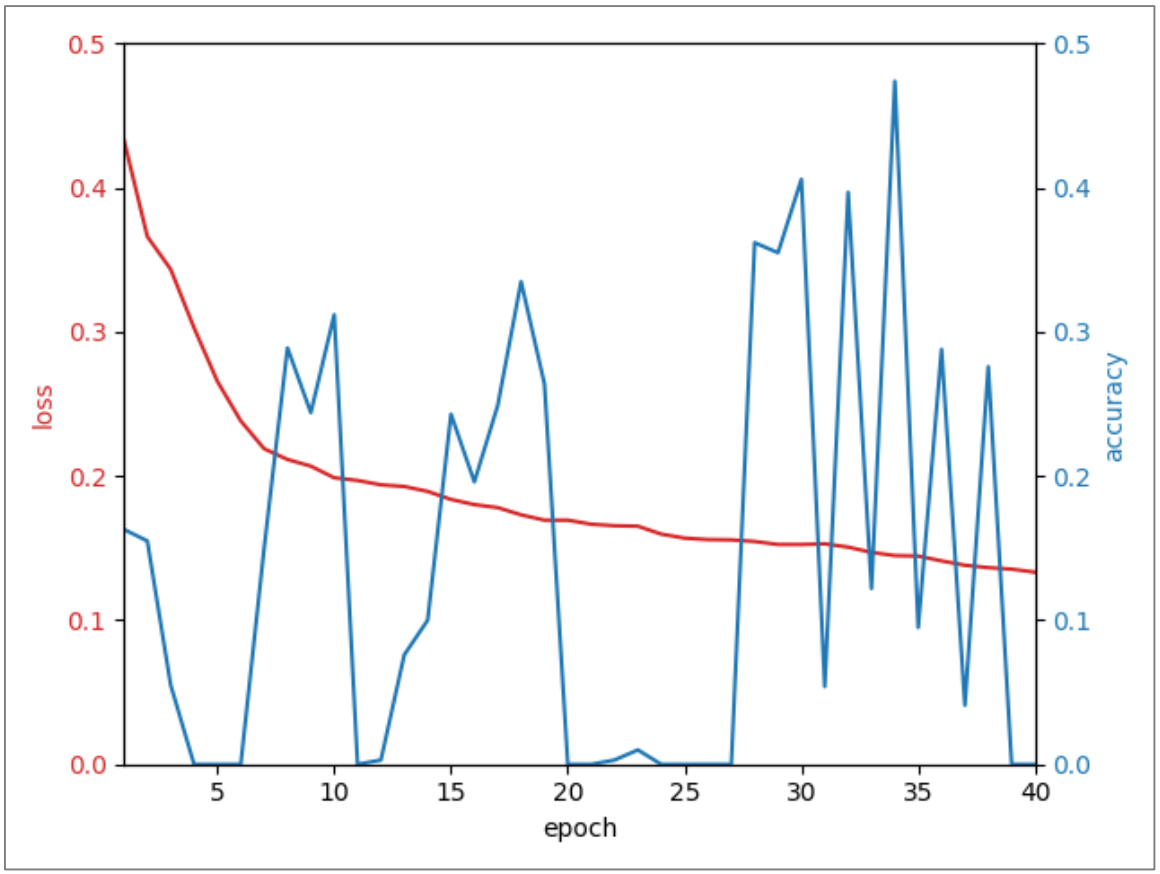
tratta di un'altra versione dei dataset *RuleTaker* D* che tratta il problema completo iterativo. In questo caso, a ciascuna teoria non sono associate le possibili inferenze, ma svariate domande a cui rispondere. Ho nuovamente esteso la classe *Dataset* [34] e ho fatto in modo che ogni domanda risulti un esempio a sé stante con le frasi della teoria casualmente riordinate. Per ciascun campione vengono restituiti teoria, domanda e risposta in formato testuale.

3.3.4. Osservazioni

La rete *T5* [12] è distribuita in varie versioni: T5-small, T5-base, T5-large, T5-3B e T5-11B, in ordine crescente di dimensioni. Con il mio setup *Google Colab*, sono in grado di eseguire l'inferenza fino alla rete T5-large e di compiere l'addestramento agevolmente solo sulla rete T5-small e molto lentamente sulla T5-base, mentre il *ProofWriter* [17] originale è basato su T5-11B, con qualche esperimento in appendice sulla T5-large.

Stando agli esperimenti che ho svolto sulla rete T5-small entro un massimo di 50 epoche d'addestramento, lo spazio delle soluzioni appare molto complesso, inteso come lo spazio di ricerca della funzione ottima che il modello neurale può apprendere. L'accuratezza del modello non cresce regolarmente e dolcemente ad ogni epoca, ma raggiunge ciclicamente dei picchi per poi tornare vicina allo zero per svariate epoche successive. L'epoca alla quale si raggiunge il miglior picco varia drasticamente in base al seme con cui si inizializzano le componenti randomiche del sistema e quando l'accuratezza torna a zero, può rimanervi anche per lungo tempo.

Riporto in figura, a titolo di esempio, un tipico grafico di addestramento del *ProofWriter* iterativo su una rete T5-small. Il valore della funzione di loss si riferisce al dataset d'addestramento, mentre il valore di accuracy si riferisce alla percentuale (tra zero e uno) di inferenze corrette ottenute sul dataset di validazione:



Andamento del tipico ciclo d'apprendimento. Figura generata traminte [37] a partire dai dati.

Stando a quanto visto, si direbbe particolarmente complicato raggiungere un addestramento completo. Con il mio setup *Google Colab*, 50 epoche di addestramento, paragonabili a circa 175˙000 step con batch di 8 esempi, impiegano generalmente più di 20 ore di tempo per essere completate e non è facile prevedere quante epoche siano necessarie a raggiungere il miglior possibile stato d'addestramento. A conferma della complessità del problema, i ricercatori stessi del *ProofWriter* riportano di aver condotto i loro esperimenti su modelli T5-11B allenati per 40˙000 step con batch di 8 esempi e che il modello con punteggio migliore in fase di validazione è spesso risultato essere l'ultimo, probabile indicazione di un addestramento incompleto.

Un'altra variabile da tenere in considerazione è certamente il formato dei dati. Dal momento che il sistema riceve e restituisce sequenze testuali, anziché dati puramente categorici, è importante evidenziare che è possibile codificare le informazioni in svariati modi differenti. Non è assolutamente scontato, infatti, che la rete sia indifferente alla scelta del formato.

Un piccolo esperimento che ho condotto ha ad esempio evidenziato che variando leggermente il formato della dimostrazione, l'accuratezza del modello migliora nettamente, almeno per quanto riguarda le prime 30 epoche d'addestramento sulla rete T5-small. Questo risultato resta consistente anche ripetendo l'esperimento con inizializzazioni differenti delle componenti casuali:

```
Formato originale:
# <regola> <fatto>
# <regola> & <fatto1> <fatto2>
Ovvero, prima la regola e poi i fatti, con eventuale "and" prefisso
Formato alternativo:
# <fatto> <regola>
# <fatto1> & <fatto2> <regola>
Ovvero, prima i fatti e poi la regola, con eventuale "and" infisso
```

La calibrazione del formato e dei parametri precedentemente descritti appare critica per i modelli relativamente piccoli e poco addestrati, ma non è certamente immediato determinare quanto risulti importante, o quali siano i valori da preferirsi, nel caso generale. Avendo a disposizione dell'hardware più performante, sarebbe interessante svolgere esperimenti più precisi in tal senso, ma una volta determinati i parametri e i formati migliori, la vera sfida rimarrebbe comunque l'interpretazione del loro significato.

Il risultato più interessante scaturito da questa analisi è tuttavia un altro. Al crescere del parametro *num_beams*, già descritto, aumentano enormemente le probabilità che tra le sequenze generate dal modello ve ne sia una corretta. Si considerino, ad esempio, i risultati di un modello T5-large fornito pre-addestrato dai ricercatori del *ProofWriter* e di un modello T5-small da me preparato, entrambi addestrati sul dataset *RuleTaker* D3 [18] per modelli iterativi e relativamente testati sul numero di inferenze corrette:

Dimensioni	Step	Risposte corrette al primo tentativo	Risposte corrette entro quattro tentativi
T5-large	≤ 40˙000	98,4 %	99,8 %
T5-small	≤ 175˙000	33,0 %	48,4 %

Si direbbe, quindi, che una fetta importante degli errori commessi dal sistema non sia imputabile all'incapacità della rete neurale di generare la risposta corretta, ma piuttosto alla sua incapacità di distinguere la sequenza corretta da quelle incorrette.

Se si riuscisse, dunque, a distinguere una risposta corretta tra le n a piacere generate dal modello, si potrebbero incrementare significativamente le prestazioni del sistema. Operando in questo modo, anche il problema già discusso dell'early-stopping si ricondurrebbe semplicemente ad un caso particolare di ricerca in cui il primo tentativo proposto dal modello è la terminazione delle iterazioni.

4. Integrazione con un nuovo approccio simbolico

Nei capitoli precedenti ci siamo domandati come poter migliorare il funzionamento di *ProofWriter* [17] e, più ingenerale, di tutte quelle architetture che impiegano una rete *T5* [12] come componente fondamentale per risolvere problemi di ragionamento.

Abbiamo visto che una tecnica piuttosto promettente, e ancora molto poco esplorata, consisterebbe nel far generare al modello neurale svariate proposte di soluzione per poi analizzarle e scegliere come risposta finale quella che risulta più promettente. Questo si direbbe, come risultato dalla precedente analisi, un ottimo punto in cui intervenire sul funzionamento del sistema con informazioni aggiuntive in campo linguistico o pertinenti al dominio del problema. Un intervento di questo tipo fornirebbe anche e soprattutto una buona base per affrontare il problema comune dell'early-stopping, altrimenti piuttosto difficile da indirizzare.

Nella sezione **Configurazione della rete T5** si descriverà nel dettaglio l'approccio generale proposto, mentre nelle sezioni **Un test di ragionevolezza** e **Un test di terminazione** si affronteranno i problemi di valutazione, rispettivamente, delle sequenze standard e delle sequenze di terminazione. Successivamente, nella sezione **Risultati ottenuti e analisi degli errori** si discuteranno i successi e le criticità delle soluzioni proposte.

4.1. Configurazione della rete T5

La funzione *HuggingFace* [35] che riceve una sequenza testuale e il modello neurale *T5* [12] attraverso cui farla passare, se opportunamente configurata, restituisce all'utente n possibili sequenze risposta, ciascuna associata al proprio punteggio di confidenza, il quale può essere facilmente convertito in un valore percentuale tramite una funzione softmax.

Riporto di seguito una porzione di codice significativa che riassume la configurazione proposta per le funzioni utilizzate, seguita da una descrizione dei parametri di input e output coinvolti:

```
model = T5ForConditionalGeneration.from_pretrained(
    path,
    local_files_only = True )
model.to('cuda')
tokenizer = T5Tokenizer.from_pretrained(
    path,
    local_files_only = True )
source = tokenizer(
    source_text,
    padding = 'max_length',
    max_length = 640,
    pad_to_max_length = True,
    truncation = True,
    return_tensors = 'pt' )
source_ids = source['input_ids'].to(dtype=torch.long)
source_mask = source['attention_mask'].to(dtype=torch.long)
ids = source_ids.to('cuda', dtype=torch.long)
mask = source_mask.to('cuda', dtype=torch.long)
generated_ids = model.generate(
    input_ids = ids,
    attention_mask = mask,
    max_length = 64,
    num_beams = nb,
    repetition_penalty = 2.5,
    length_penalty = 0.5,
    early_stopping = False,
    num_return_sequences = nb,
    output_scores = True,
    return_dict_in_generate = True )
predictions = [
    tokenizer.decode(
        g,
        skip_special_tokens = True,
        clean_up_tokenization_spaces = True )
    for g in generated_ids.sequences ]
scores = torch.softmax(
    generated_ids.sequences_scores,
    dim = -1 ).tolist()
```

***path*:** stringa in input che rappresenta il percorso alla cartella contenente i file del modello e del relativo tokenizer.

***source_text*:** stringa in input contenente la sequenza da fornire alla rete.

***nb*:** numero intero in input che rappresenta il numero di sequenze alternative che la rete deve generare in risposta alla sequenza *source_text*.

***predictions*:** lista di stringhe in output contenente le *nb* sequenze generate dal modello.

***scores*:** lista di numeri reali in output che rappresenta la percentuale di confidenza associata a ciascuna sequenza contenuta nella lista *predictions*; la somma di questi punteggi ha valore 1, ovvero 100%.

Assumendo poi, seppur impropriamente, che vi sia sempre almeno una sequenza corretta tra quelle generate, un approccio vincente potrebbe consistere nell'esaminare le sequenze in ordine calante di confidenza e scegliere come risposta definitiva la prima che risulti ragionevole secondo una certa euristica ancora da definire.

Un caso particolare, abbastanza singolare da poter essere considerato come sotto-problema a sé stante, si ha quando la rete produce la sequenza che indica che non vi sarebbero ulteriori inferenze derivanti dalla teoria. Il confronto tra tale sequenza, che non può essere intrinsecamente irragionevole, e le altre sequenze ragionevoli risulta infatti piuttosto complicato. Questo perché, a differenza delle sequenze standard che propongono delle coppie risposta-dimostrazione piuttosto informative, la sequenza di terminazione non è in grado di fornire informazioni aggiuntive.

4.2. Un test di ragionevolezza

Una caratteristica tipica delle reti neurali è il fatto che quando forniscono una risposta errata, esse commettono spesso errori in maniera apparentemente illogica. Questo difetto può, in questo caso, essere sfruttato a nostro vantaggio. Molte sequenze incorrette possono infatti essere scartate tramite alcune semplici verifiche formali.

Il primo controllo predisposto si assicura, infatti, che le sequenze soddisfino le seguenti caratteristiche: (1) ogni etichetta dichiarata nella dimostrazione deve effettivamente essere presente all'interno della teoria; (2) la nuova inferenza prodotta non deve essere già presente all'interno della teoria; (3) ciascuna frase della teoria può essere usata al massimo una volta all'interno dello stesso passaggio di una dimostrazione. È infatti impossibile, almeno nel caso del dataset *RuleTaker* [18], che una stessa frase sia usata correttamente più volte per dimostrare lo stesso step d'inferenza, mentre *ProofWriter* [17] commette spesso errori di ripetizione. In *RuleTaker* le regole hanno solo due possibili forme: (1) un fatto implica un nuovo fatto o (2) la congiunzione di due fatti implica un nuovo fatto, dunque una ripetizione indicherebbe un fatto impiegato incorrettamente come regola o viceversa oppure un fatto congiunto con sé stesso, caso formalmente accettabile, ma non contemplato dalle regole in quanto degenere.

L'errata ripetizione delle etichette è una fonte d'errore piuttosto frequente, assieme alla confusione tra etichette lessicalmente simili, come ad esempio *sent1* e *sent11*. Al di là dello specifico metodo impiegato, resta opportuno in ogni caso trovare un modo per arginare questi problemi.

Superato il primo step di correttezza formale, ho predisposto un passaggio successivo volto a misurare la plausibilità delle sequenze. Potremmo dire, a grandi linee, che una sequenza è tanto più plausibile quanto più le frasi coinvolte sono legate da argomenti in comune. In termini di codice, ho nuovamente impiegato la libreria *spaCy* [24] per suddividere il testo della risposta candidata e quello della regola e dei fatti citati nella dimostrazione in token in forma base. Si può dunque verificare la validità del legame tra le proposizioni assicurandosi che il testo della regola, punto di collegamento tra tutte le frasi, abbia in comune con il testo delle altre almeno un certo numero di token rilevanti.

La regola per distinguere quali token siano rilevanti e il numero minimo di corrispondenze necessarie a segnalare una risposta come plausibile sono da considerarsi parametri del sistema e devono essere opportunamente tarati.

Prendendo in considerazione le caratteristiche specifiche del dataset *RuleTaker*, ma mantenendo un approccio sufficientemente generale, ho deciso di considerare come rilevanti solamente i token contrassegnati da *spaCy* come aggettivi o verbi non ausiliari, in modo tale da scartare quelle parole che sono molto comuni e non indicano particolari collegamenti logici, come tipico ad esempio di articoli e preposizioni. Ho scelto, almeno per questo prototipo, di scartare anche tutti gli avverbi, poiché pur essendovene molti distintivi, ve ne sono anche tanti fuorvianti. Ho deciso di scartare, allo stesso modo, anche i nomi, poiché vengono spesso usati l'uno al posto dell'altro in maniera poco distintiva. Si pensi, ad esempio, a come il termine *someone* e il termine *Adam* possano riferirsi allo stesso concetto.

A motivo di questa restrizione alle categorie lessicali considerate, ho dovuto, per quanto riguarda la soglia minima, accettare come plausibili tutte le situazioni in cui è presente almeno una corrispondenza tra la regola e ciascuna altra frase coinvolta.

Il seguente esempio illustra un caso tipico del procedimento descritto, in cui si può anche notare la necessità di una soglia così bassa:

```
Teoria:
sent1: The cat chases the squirrel.
sent2: If someone is cold then they are rough.
sent3: If the bald eagle is big and the bald eagle chases the cat
       then the cat is rough.
sent4: If someone is rough then they like the cat.
sent5: If someone chases the squirrel then the squirrel is rough.
sent6: The squirrel chases the cat.
sent7: If someone sees the bald eagle and they chase the bald eagle
       then the bald eagle chases the squirrel.
sent8: If someone likes the bald eagle and they see the squirrel then
       the bald eagle is rough.
sent9: The squirrel is rough.

Inferenza:                      Dimostrazione:
The squirrel likes the cat.     # sent4 sent9

Ragionevolezza formale:
1. Le etichette sent4 e sent9 fanno parte della teoria? OK
2. La frase “The squirrel likes the cat.” è nuova? OK
3. Le etichette sent4 e sent9 sono non ripetute nella dim.? OK

Calcolo plausibilità:
Token significativi nella regola (sent4): {rough, like}
Token significativi nel fatto (sent9): {rough} OK
Token significativi nella risposta (inferenza): {like} OK
```

Un’ulteriore motivazione a supporto della scelta di tale soglia minima è data dal fatto che *spaCy* non sempre classifica correttamente le parole, riducendo così ulteriormente il pool utile alla valutazione.

Si accetta, in questo modo, come risposta corretta la sequenza plausibile con valore di confidenza più elevato. Ho anche preso in considerazione la possibilità di ignorare il punteggio di confidenza e selezionare semplicemente la sequenza con il maggior numero di corrispondenze, ma all’atto pratico si è dimostrata una strategia meno efficace.

4.3. Un test di terminazione

Il test per determinare se la richiesta da parte del modello di terminare le iterazioni sia ragionevole o meno è decisamente più insidioso rispetto

al caso precedente poiché non vi sono categorie lessicali altrettanto semplici da sfruttare.

La mia proposta, come possibile euristica da esplorare, è la seguente: se, in ordine calante di confidenza, la prima sequenza ragionevole incontrata è una richiesta di terminazione, procediamo anzitutto verificando che la sequenza successiva risulti anch'essa ragionevole. In tal caso si passa ad un confronto dettagliato, altrimenti, se la sequenza successiva non è ragionevole o è del tutto assente, si seleziona la terminazione come risposta corretta.

Il problema del confronto dettagliato è un problema complesso poiché è necessario confrontare una richiesta di terminazione, lessicalmente sempre identica e priva di dimostrazione, con una coppia risposta-dimostrazione già giudicata ragionevole.

I dati utilizzabili per il confronto sono la percentuale di confidenza della richiesta di terminazione, la percentuale di confidenza della nuova inferenza proposta ed il grado di plausibilità di tale inferenza espresso, come descritto nella sezione precedente, dal numero di token base in comune tra la regola utilizzata e ciascuna delle altre frasi coinvolte, inclusa la risposta stessa.

Risulta subito evidente che queste informazioni non siano in grado di fornire una risposta conclusiva e definitiva, tuttavia potrebbero essere sufficienti, mediante uno studio approfondito, a produrre una regola generale in grado di stimare con una certa precisione la scelta corretta, contribuendo così a mitigare il problema dell'early-stopping introdotto nei capitoli precedenti. Ci si potrebbe affidare, ad esempio, ad un nuovo modello neurale addestrato allo scopo. Ho tuttavia scartato questa possibilità, ai fini del prototipo, poiché un focus fondamentale di questo lavoro è la riduzione del grado di incertezza del sistema e procedere in questa direzione diminuirebbe significativamente l'interpretabilità dell'architettura.

Un modo efficace per sperimentare un gran numero di semplici test consiste, invece, nell'affidarsi a quei modelli di machine-learning che

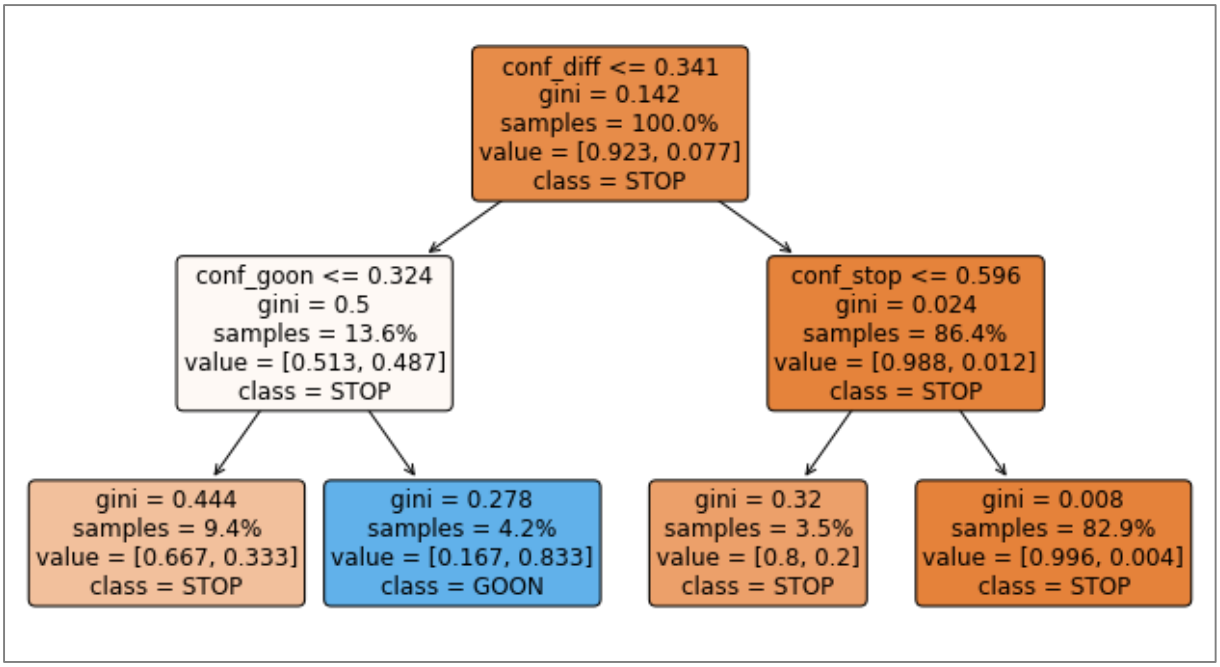
hanno un funzionamento più interpretabile, in particolare agli alberi decisionali. Queste strutture sono equivalenti, a livello funzionale, a delle catene di blocchi if-else. L'ordine in cui prendere in considerazione variabili e valori viene però, in questo modo, determinato automaticamente a partire dai dati.

Ho dunque impiegato libreria Python *Scikit-learn* [38] predisponendo un albero decisionale che prendesse in input i seguenti valori: (1) confidenza della richiesta di terminazione, (2) confidenza della successiva inferenza proposta, (3) corrispondenze tra regola e risposta, (4) corrispondenze tra regola e primo fatto, (5) corrispondenze tra regola e secondo fatto (se presente). Ho poi incluso come dato derivato anche la (6) differenza tra le due confidenze.

Il ciclo d'addestramento di tale albero accetta un dataset modellato come i precedenti e sfrutta lo stesso modello *T5* [12] per generare le sequenze e filtrarle, mantenendo solo quegli esempi che si trovano nella situazione critica che richiede un test di terminazione. Tali esempi vengono poi impiegati, in cross-validation, anche per tarare i meta-parametri dell'albero, in particolare la sua profondità massima.

Per addestrare l'albero ho impiegato unicamente il dataset usato per la validazione, dal momento che il modello potrebbe avere un forte bias sul dataset d'addestramento e che impiegare il dataset di test, o una sua parte, per la taratura dell'albero complicherebbe il confronto con i test precedenti. Ho comunque effettuato alcuni esperimenti accessori utilizzando gli altri dataset e ho potuto verificare che vengono ugualmente prodotti alberi molto simili.

In ogni caso, questi alberi tendono tipicamente ad impiegare un numero ridotto di confronti e a concentrarsi sui valori di confidenza, ignorando i dati legati alla plausibilità. Si osservi, ad esempio, l'albero decisionale riportato in figura:



Albero addestrato al test di terminazione. Figura generata tramite [38] a partire dai dati.

In questo caso, il test suggerisce al sistema di ignorare la richiesta di terminazione solo quando la differenza tra le confidenze è non superiore al valore 0,341 e, contemporaneamente, la confidenza della sequenza di non-terminazione è superiore al valore 0,324. Per interpretare il significato di tali valori numerici, si consideri che i punteggi di confidenza sono numeri reali positivi organizzati in modo tale che la somma delle confidenze delle n sequenze generate per ciascun esempio abbia somma 1, con il significato di 100%. Dunque, secondo l'euristica espressa da questo albero, se la confidenza di non-terminazione supera il 32,4% e la confidenza di terminazione non supera il 66,5% (0,324 + 0,341) allora le iterazioni proseguono, altrimenti terminano.

4.4. Risultati ottenuti e analisi degli errori

Riporto di seguito i risultati ottenuti impiegando i test descritti nelle sezioni precedenti. I dati si riferiscono a modelli addestrati e testati sul dataset *RuleTaker* D3 [18] in OWA per modelli iterativi. L'accuratezza è intesa come la percentuale di volte in cui è stata selezionata la risposta esatta (inclusa dimostrazione corretta). Per ciascun esempio sono state fatte generare al modello $n = 4$ possibili sequenze, il numero massimo gestibile con il mio setup *Google Colab* descritto in introduzione.

Accuratezza precedente: Accuratezza ottenuta seguendo il paradigma originale del *ProofWriter* iterativo [17].

Nuova accuratezza: Accuratezza ottenuta impiegando i test di ragionevolezza e di terminazione proposti in questo capitolo.

Accuratezza con oracolo: Accuratezza teoricamente raggiungibile se fossimo sempre in grado di distinguere una sequenza corretta tra quelle prodotte dal modello, ovvero, la percentuale di volte in cui il modello ha fornito almeno una sequenza corretta tra le $n = 4$ generate.

Dimensioni	Step	Accuratezza precedente	Nuova accuratezza	Accuratezza con oracolo
T5-large	≤ 40˙000	98,4 %	99,1 %	99,8 %
T5-small	≤ 175˙000	33,0 %	43,1 %	48,4 %

Si può notare come la soluzione proposta permetta di identificare e correggere una porzione notevole degli errori precedentemente commessi dalla rete neurale.

Analizzando nel dettaglio gli errori commessi durante il nuovo test dal miglior modello disponibile, ovvero la rete T5-large fornita dagli autori di *ProofWriter*, emerge la seguente distribuzione:

Causa d'errore	Frequenza
inferenza corretta immediatamente preceduta da inferenza ragionevole scorretta	37,7 %
early-stopping	14,5 %
nessuna inferenza corretta generata dal modello	11,6 %
inferenza corretta indirettamente preceduta da inferenza ragionevole scorretta	10,1 %
errata correzione di un falso positivo di early-stopping	8,7 %
terminazione corretta immediatamente preceduta da inferenza ragionevole scorretta	8,7 %
inferenza corretta scartata poiché irragionevole	4,3 %
terminazione corretta indirettamente preceduta da inferenza ragionevole scorretta	4,3 %

Circa il 23% degli errori rimanenti risulta dunque imputabile al modulo di verifica della terminazione, mentre oltre il 65% degli errori sfuggiti alla correzione sarebbe, più o meno direttamente, di competenza del modulo di valutazione della ragionevolezza. Risulta dunque evidente che il principale componente della soluzione proposta da rivisitare nell'ottica di un miglioramento del prototipo sarebbe quest'ultimo.

A conclusione dei risultati già ottenuti, sarebbe opportuno effettuare un test esaustivo sul dataset *RuleTaker* D3 OWA nella versione che tratta il problema completo, ovvero la generazione libera da parte del modello delle inferenze, seguita dalla ricerca della risposta al quesito posto. Sfortunatamente, il test sull'intero dataset non è affrontabile, in termini di tempo, con il mio setup *Google Colab*. Presento quindi di seguito i risultati ottenuti testando il sistema su un piccolo campione di 3˙600 elementi estratti casualmente:

Dimensioni	Step	Accuratezza precedente	Nuova accuratezza
T5-large	≤ 40˙000	99,1 %	99,2 %
T5-small	≤ 175˙000	72,7 %	75,7 %

Il test nei quattro casi riportati ha richiesto, complessivamente, circa 20 ore di calcolo. Basandosi su questo piccolo campione, non necessariamente esaustivo, si confermano i risultati riportati dai test precedenti.

5. Conclusioni

Nella sezione **Conclusioni nel merito di ProofWriter** si presenteranno i risultati ottenuti nel merito dello specifico caso di studio, mentre nella sezione **Conclusioni nel merito delle reti T5** si riassumerà la metodologia generale precedentemente approfondita e applicabile su più vasta scala. Nella sezione **Possibili sviluppi futuri** si riporteranno i principali punti lasciati aperti durante questa trattazione.

Il codice a supporto della tesi è disponibile online al seguente indirizzo: `https://github.com/JackFantaz/T5-Beams-Exploration-for-Reasoning.git`

5.1. Conclusioni nel merito di ProofWriter

Nel capitolo precedente, ho mostrato come si possa estendere *ProofWriter* [17] tramite tecniche completamente interpretabili al fine di migliorare la correttezza delle risposte prodotte. Un'attenta analisi delle sequenze generate dal modello, mediante opportune euristiche, permette infatti di contrastare le cause d'errore più comuni e di incrementare significativamente le prestazioni del modello. Si ottiene in questo modo un beneficio netto in fase di test, pur introducendo alcuni nuovi errori.

5.2. Conclusioni nel merito delle reti T5

Sfruttando la funzione *HuggingFace* [35] che facilita la generazione di più sequenze alternative tramite beam-search, si è in grado di far generare ai modelli neurali della famiglia *T5* [12] più possibili risposte ad una certa sequenza in ingresso, per poi procedere ad una selezione della risposta definitiva tramite tecniche esterne alla rete. Si può quindi demandare ad un modulo sub-simbolico opportunamente addestrato il compito di proporre possibili soluzioni ad un certo problema e configurare il resto del sistema come un verificatore di tali proposte. La modellazione di un problema può in questo modo essere ricondotta al solo riconoscimento, spesso più semplice, di istanze corrette della sua soluzione.

5.3. Possibili sviluppi futuri

In primo luogo, si potrebbe verificare il funzionamento dell'estensione proposta di *ProofWriter* su split di *RuleTaker* [18] differenti da D3 o addirittura adattarne il funzionamento ed effettuare test su altri dataset simili, come *EntailmentBank* [25]. Bisognerebbe poi verificare la capacità teorizzata della metodologia proposta di adattarsi agli altri sistemi basati su modelli neurali *T5* [12], come *NLProofS* [21] e *FaiRR* [23].

In secondo luogo, si potrebbe riprendere il prototipo proposto di *ProofWriter* esteso e tentare di migliorarlo ideando e implementando funzioni euristiche più articolate, come accennato in fase di analisi degli errori, sperimentando anche la possibilità di impiegare a questo scopo ulteriori modelli sub-simbolici. Altre configurazioni importanti, e al momento tralasciate, sono la scelta del formato con cui fornire e richiedere i dati alla rete, che, come precedentemente spiegato, potrebbe avere un impatto notevole sul corretto funzionamento del sistema, e la calibrazione dei parametri *repetition_penalty* e *length_penalty* della funzione di generazione delle sequenze.

In terzo luogo, sarebbe opportuno ripetere tutti gli esperimenti effettuati su *ProofWriter* senza le limitazioni imposte dall'hardware, dunque impiegando una rete T5-11B, come nello studio originale [17], e mettendo alla prova il prototipo sull'intero dataset di *RuleTaker* riferito al problema in forma completa.

Si potrebbero, infine, investigare più nel dettaglio i vari problemi aggiuntivi sollevati in fase di analisi del problema, come l'individuazione del contrario di una certa affermazione o la messa a punto di un confronto conclusivo tra le architetture proposte.

Riferimenti

[1] J. McCarthy, "Programs with Common Sense", *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, 1959.

[2] S. J. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach, 4th Edition*, Pearson Education, 2021.

[3] F. C. N. Pereira, S. M. Shieber, *Prolog and Natural-Language Analysis*, Center for the Study of Language and Information, 1987.

[4] L. Weber, P. Minervini, J. Münchmeyer, U. Leser, T. Rocktäschel, "NLProlog: Reasoning with Weak Unification for Question Answering in Natural Language", *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2019.

[5] A. Saparov, T. M. Mitchell, "Towards General Natural Language Understanding with Probabilistic Worldbuilding", *Transactions of the Association for Computational Linguistics*, 2022.

[6] R. Kowalski, "Logical English", *Logic and Practice of Programming*, 2020.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, "Attention Is All You Need", *Advances in Neural Information Processing Systems*, 2017.

[8] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding", *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

[9] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, "Improving Language Understanding by Generative Pre-Training", *OpenAI Technical report*, 2018.

[10] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach", *arXiv preprint*, 2019.

[11] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, R. Soricut, "ALBERT: A Lite BERT for Self-Supervised Learning of Language Representations", *International Conference on Learning Representations*, 2020.

[12] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer", *Journal of Machine Learning Research*, 2020.

[13] S. Narang, C. Raffel, K. J. Lee, A. Roberts, N. Fiedel, K. Malkan, "WT5?! Training Text-to-Text Models to Explain their Predictions", *arXiv preprint*, 2020.

[14] C. Helwe, C. Clavel, F. Suchanek, "Reasoning with Transformer-Based Models: Deep Learning, but Shallow Reasoning", *Automated Knowledge Base Construction*, 2021.

[15] M. Hahn, "Theoretical Limitations of Self-Attention in Neural Sequence Models", *Transactions of the Association for Computational Linguistics*, 2020.

[16] S. Bhattamishra, K. Ahuja, N. Goyal, "On the Practical Ability of Recurrent Neural Networks to Recognize Hierarchical Languages", *International Conference on Computational Linguistics*, 2020.

[17] O. Tafjord, B. Dalvi, P. Clark, "ProofWriter: Generating Implications, Proofs, and Abductive Statements over Natural Language", *Findings of the Association for Computational Linguistics*, 2021.

[18] P. Clark, O. Tafjord, K. Richardson, "Transformers as Soft Reasoners over Language", *Proceedings of the International Joint Conference on Artificial Intelligence*, 2020.

[19] S. Saha, S. Ghosh, S. Srivastava, M. Bansal, "PRover: Proof Generation for Interpretable Reasoning over Rules", *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2020.

[20] K. Yang, J. Deng, "Learning Symbolic Rules for Reasoning in Quasi-Natural Language", *arXiv preprint*, 2021.

[21] K. Yang, J. Deng, C. Danqi, "Generating Natural Language Proofs with Verifier-Guided Search", *arXiv preprint*, 2022.

[22] K. Bostrom, Z. Sprague, S. Chaudhuri, G. Durrett, "Natural Language Deduction through Search over Statement Compositions", *arXiv preprint*, 2022.

[23] S. Sanyal, H. Singh, X. Ren, "FaiRR: Faithful and Robust Deductive Reasoning over Natural Language", *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2022.

[24] M. Honnibal, I. Montani, "spaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks and Incremental Parsing", *To appear*, 2017.

[25] B. Dalvi, P. Jansen, O. Tafjord, Z. Xie, H. Smith, L. Pipatanangkura, P. Clark, "Explaining Answers with Entailment Trees", *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2021.

[26] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, O. Tafjord, "Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge", *arXiv preprint*, 2018.

[27] Z. Xie, S. Thiem, J. Martin, E. Wainwright, S. Marmorstein, P. Jansen, "WorldTree v2: A Corpus of Science-Domain Structured Explanations and Inference Patterns supporting Multi-Hop Inference", *Proceedings of the Language Resources and Evaluation Conference*, 2020.

[28] P. Jansen, E. Wainwright, S. Marmorstein, C. Morrison, "WorldTree: A Corpus of Explanation Graphs for Elementary Science Questions Supporting Multi-Hop Inference", *Proceedings of the International Conference on Language Resources and Evaluation*, 2018.

[29] S. R. Bowman, G. Angeli, C. Potts, C. D. Manning, "A Large Annotated Corpus for Learning Natural Language Inference", *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2015.

[30] O.-M. Camburu, T. Rocktäschel, T. Lukasiewicz, P. Blunsom, "e-SNLI: Natural Language Inference with Natural Language Explanations", *Advances in Neural Information Processing Systems*, 2018.

[31] N. F. Rajani, B. McCann, C. Xiong, R. Socher, "Explain Yourself! Leveraging Language Models for Commonsense Reasoning", *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2019.

[32] A. Talmor, J. Herzig, N. Lourie, J. Berant, "CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge", *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

[33] T. Sellam, D. Das, A. P. Parikh, "BLEURT: Learning Robust Metrics for Text Generation", *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2020.

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library", *Advances in Neural Information Processing Systems*, 2019.

[35] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, A. M. Rush, "Transformers: State-of-the-Art Natural Language Processing", *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2020.

[36] N. M. Shazeer, M. Stern, "Adafactor: Adaptive Learning Rates with Sublinear Memory Cost", *arXiv preprint*, 2018.

[37] J. D. Hunter, "Matplotlib: A 2D Graphics Environment", *Computing in Science and Engineering*, 2007.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, 2011.