Natural Language Processing

appunti e note raccolte 2023/2024

Dagli appunti di: Giulio Appetito, Valerio Brauzi, Anastasia Brinati, Andrea Cantarini, Paolo Ciasco, Luca Corridori

Indice

*

Introduzione

Lo scopo di questo documento è raccogliere appunti e note degli studenti del corso NLP, e creare un riassunto sostanzioso e fruibile a chiunque negli anni a venire, con la speranza che venga ampliato, aggiornato e modificato da futuri studenti.

"What is natural language and what do we want to do with it?",
è la domanda di apertura del corso. Lo scopo del linguaggio è far comunicare due (o più) entità.
Secondo *Ferdinand De Saussure* i parlanti si accordano per chiamare oggetti e concetti in determinate maniere. La ragion d'esistere delle parole non c'è, l'accezione è relativa ad un aspetto sociale, poichè il linguaggio è un fenomeno dovuto alla socialità. Parliamo infatti di linguistica **esterna**, ovvero studio del linguaggio come entità esterna all'individuo. Distinguiamo inoltre la **langue**, l'entità sociale propria di una comunità, canonizzata e dunque indrottinata ai cervelli, dalla **parole**, che è invece l'entità naturale, espressione con la propria cognizione (=attuazione individuale) della langue.
Chomsky tratta invece della linguistica interna, studio del linguaggio come capacità cognitiva dell'individuo, dove la distinzione è fra **competence**, capacità d apprendere, e **performance**, capacità di produzione del linguaggio. I large language models lavorano sulla parole, anche se inizialmente l'NLP, le discipline Corpus Linguistics e gli EMNLP (empirical metaods for NLP), si concentravano sulla competence. Anche se la parole evolve continuamente, la langue stessa non è stabile: le parole nascono, muoiono e cambiano di significato.

Parte 1

CAPITOLO 1

Introduzione al fenomeno della lingua

1.1 Significato e significante

Il linguaggio è uno strumento per trasmettere informazioni attraverso una serie di simboli. Ma cosa è un simbolo? Dato un oggetto (significato), un simbolo è ciò che lo rappresenta (significante). I veicoli segnici rappresentano i sensi delle parole, che sono nell'iperuranio e nella testa dei parlanti.

Tuttavi andare dal veicolo segnico al senso e viceversa non è immediato poichè la relazione è 'molti a molti', infatti, la rappresentazione naturale soffre di molti "difetti", come ad esempio la ricchezza espressiva (più veicoli segnici vanno in un unico senso), l'arbitrarietà e l'ambiguità (più sensi vanno in uno stesso veicolo segnico). Esistono diversi tipi di ambiguità a seconda del livello linguistico in cui ci troviamo.

Osservazione 1.1.1 Esempio: Una vecchia legge la regola

1.2 Livelli linguistici

Fonologia: branca della linguistica che studia i sistemi di suoni; la più semplice particella che studia la fonologia è il *fonema*, che può essere un singolo suono.
Morfologia: è lo studio delle parole, la loro composizione, e l'appartenenza a determinate categorie (nome, verbo, etc...). Nella linguistica moderna studia la struttura della parola e descrive le varie forme che assume a seconda delle categorie di numero, genere, modo, tempo e persona. Le parole sono costituite di *morfemi*, unità minime con significato autonomo, ..
Sintassi: branca della grammatica e linguistica che studia i diversi modi (relazioni) in cui i codici dei linguaggi si uniscono fra loro per formare una preposizione, "unità superiori alla parola".
Semantica: studio del *significato delle parole* (lessicale), e delle frasi (frasale), considerando il rapporto fra l'espressione e la realtà extralinguistica. Il significato di una frase è dato da quello delle singole parti, più quello degli eventuali elementi connettori (anche se non è detto che ciò sia sufficiente, considerando ad esempio metafore o catacresi).
Pragmatica: studio del linguaggio in rapporto all'uso contestuale che ne fa il parlante, come il contesto contribuisce al significato.

1.3 Analisi Morfemica

Questa analisi, detta "analisi morfemica", consiste nell'individuazione di confini tra elementi linguistici e avviene tramite segmentazione. Nella linguistica descrittiva moderna, il **morfema** è il più piccolo elemento dotato di significato in una parola (o in un enunciato) e si colloca sul piano della prima articolazione. In questo senso, rappresenta l'**unità di analisi** della morfologia.

Se prendiamo ad esempio le parole italiane "parla", parlavo, parlando, parlante, parlare, è facile identificare e distinguere l'elemento parl- e gli altri elementi (-a, -avo, -ando, -ante, -are). Tutti sono morfemi ed hanno un significato proprio (spesso indicato tra parentesi graffe). Il primo morfema (parl-) veicola il significato centrale della forma linguistica in questione. Invece, il morfema -a significa {INDICATIVO PRESENTE 3a PERSONA}; oppure, il morfema -avo significa {INDICATIVO IMPERFETTO 1a PERSONA}; il morfema -ando significa {GERUNDIO PRESENTE}; il morfema -ante significa {PARTICIPIO PRESENTE SINGOLARE}; il morfema -are significa {INFINITO}. Come si può vedere, un morfema può veicolare sincreticamente più informazioni grammaticali, ad esempio il morfema -i, in ragazzi, significa contemporaneamente {MASCHILE} e {PLURALE}.

I morfemi si suddividono in chiusi (grammaticali), che permettono di cambiare il gruppo sintattico, e aperti (lessicali), che portano il significato:

Il triangolo semiotico di Peirce.

Figura 1.1. Triangolo di Peirce

- **Morfemi Chiusi**: Sono morfemi che non possono apparire da soli come parole indipendenti. Si trovano spesso alla fine di una parola. Hanno spesso un significato grammaticale piuttosto che un significato lessicale.

  Osservazione 1.3.1 Esempio: i suffissi -ed, -ing, -s in inglese (come in walked, walking, cats)

- **Morfemi Aperti**: Sono morfemi che possono apparire come parole indipendenti. Sono spesso parole principali con significato lessicale.

  Osservazione 1.3.2 Esempio: nella parola "bookshop," "book" è un morfema aperto che può esistere come parola indipendente, mentre "-shop" è un morfema chiuso che fornisce un significato aggiuntivo ma non esiste da solo.

Osservazione 1.3.3 Esempio: (closed, prefix) re - (open) play - (closed, suffix) ing/ed/er

1.4 Interpreting Natural Language

Abbiamo stabilito che lo scopo de linguggio naturale è la comunicazione, trasmettere idee sul mondo interiore o esteriore, credenze, knowledge, concetti, emozioni... ma come lo interpretiamo? Dati dei concetti e le relazioni fra essi vogliamo ricavare istanze, fatti accaduti, (ad esempio il task: retriving information from text collection). Vogliamo definire un meccanismo (finito) che possa generare ed interpretare un linguaggio:

- interpretare: da una frase costruire il grafo(albero) di derivazione;
- generare: dal grafo ricostruire la frase.

1.5 Composizionalità sematica

Seguendo l'idea di Chomsky per cui la sintassi guida l'interpretazione semantica, il nostro scopo è costruire un modello in grado di produrre *suitable syntactic structures* delle frasi in input. Un primo approccio è stato quello di portare le frasi verso un linguaggio **logico**, ovvero un linguaggio su cui si possono fare inferenze (restrizione del linguaggio naturale).
Per la filosofia del linguaggio, il significato di una frase (espressione linguistica) è dato solamente dalle parole (espressioni costituenti) ed dagli elementi di raccordo (modalità di combinazione); immaginiamo di star trattando espressioni aritmetiche. Questo principio funziona nell'ambito della logica, ma nel linguaggio naturale non sempre, sembra che "il significato sia qualcosa di più della somma delle parti" (vedi metafore etc...). amsmath

CAPITOLO 2

Rappresentazioni dei simboli in vettori

2.1 Introduzione

Che tipo di veicoli segnici abbiamo? Il linguaggio naturale è una rappresentazione simbolica della conoscenza umana, e la composizione di simboli in parole e delle parole in frasi segue le regole che l'ascoltatore e il parlante conoscono. Una buona parte della rappresentazione è arbitraria, dipende da un accordo sociale; vediamo ad esempio i fenomeni 'ricchezza espressiva' (un unico senso può essere espresso con tanti veicoli segnici) e 'ambiguità' (un unico veicolo segnico può avere più sensi: "La borsa di pelle di nonna").

Ad oggi, con l'avanzamento del ML, i simboli discreti stanno scomparendo per essere sostituiti da vettori e tensori: si hanno dunque le cosiddette **rappresentationi distribuite o distribuzionali** (ad esempio word embeddings), in cui parole e frasi sono rappresentate come vettori o tensori di numeri reali, e i simboli discreti sopravvivono solo come input e output degli algoritmi.

> In una definizione più rigorosa il **word embedding** è un termine complessivo che indica, nell'elaborazione del linguaggio naturale, un insieme di tecniche di modellazione in cui parole o frasi di un vocabolario vengono mappate in vettori di numeri reali. Concettualmente consiste in un'operazione matematica di immersione in conseguenza della quale uno spazio costituito da una dimensione per parola viene trasformato in uno spazio vettoriale continuo di dimensione molto inferiore. Queste tecniche trovano applicazione nello studio della vicinanza semantica del discorso, in particolare nel mondo della semantica distribuzionale.

2.2 Johnson–Lindenstrauss lemma

Problema della riduzione della dimensione

It's a problem concerning low-distortion embeddings of points from high-dimensional into low-dimensional Euclidean space. The lemma states that a set of points in a high-dimensional space can be embedded into a space of much lower dimension in such a way that distances between the points are nearly preserved. In the classical proof of the lemma, the embedding is a random orthogonal projection.

Tuttavia vi è un legame tra le rappresentazioni distribuite e i simboli, in quanto la prima è una approssimazione dei secondi.

2.3 Rappresentazioni Distribuite e Simboliche: Interpretabilità e Composizionalità Concatenativa

Le **rappresentazioni distribuite** portano le espressioni simboliche in spazi metrici dove la similarità tra esempi è usata per apprendere regolarità usando algoritmi di machine learning. Date due espressioni simboliche dunque, la loro rappresentazione distribuita dovrebbe cogliere la loro similarità e alcune loro feature. La rappresentazione distribuita in NLP (Natural Language Processing) si chiama così perché coinvolge la distribuzione delle informazioni su più unità o dimensioni. In questo contesto, "distribuito" si riferisce al modo in cui le caratteristiche o i concetti sono rappresentati attraverso un insieme di parametri o vettori piuttosto che essere concentrati in un'unica posizione.

Esempio. Consideriamo due frasi, *s1="un topo mangia il formaggio"* ed *s2="Un gatto mangia il topo"*, ci sono varie similarità, come numero di parole in comune e realizzazione del patter ANIMALE MANGIA CIBO. Il punto chiave è scegliere o far scegliere a un algoritmo quale sia la migliore rappresentazione per un task specifico.

Nonostante le rappresentazioni distribuite stiano rimpiazzando le rappresentazioni simboliche discrete, queste sono meno interpretabili dagli umani, dunque dovremmo considerare alcune delle proprietà delle rappresentazioni simboliche potrebbero tornare utili.

Innanzitutto le rappresentazioni simboliche discrete sono interpretabili dagli umani perché **i simboli non sono alterati nelle espressioni**: un insieme infinito di espressioni può essere ottenuto concatenando un **insieme finito di simboli di base** in accordo a delle regole di concatenazione, processo durante il quale i simboli non vengono alterati. Usando il principio della **composizionalità semantica**, il significato delle espressioni può essere ottenuto combinando il significato delle varie parti, e ricorsivamente combinando il significato dell'insieme finito di simboli di base.

Esempio. Sia un insieme di simboli di base *D=mouse,cat,a,swallows,(,)*, allora plausibili espressioni potrebbero essere del tipo:

- s_1="a cat swallows a mouse"
- t_1=((a cat)(swallows(a mouse)))

dove la seconda è una rappresentazione strutturata ad albero in forma parentetica.

Le rappresentazioni distribuite però sembrano alterare i simboli quando vengono applicate a input simbolici, e dunque sono meno interpretabili, dal momento che i simboli vengono convertiti in vettori o tensori, che a loro volta vengono trasformati con moltiplicazioni tra matrici o con funzioni non lineari. Sorge dunque il dubbio di quale sia la relazione tra i simboli iniziali e le espressioni delle loro rappresentazioni distribuite, e come queste espressioni vengano modificate durante le operazioni tra matrici o con funzioni nonlineari.
La domanda che ci poniamo è dunque se le rappresentazioni distribuite e quelle simboliche siano diverse per via dell'alterazione dei simboli. Per contribuire alla risposta, *Gelder(1990)* ha formalizzato questa proprietà di alterare i simboli con due nozioni di composizionalità: **composizionalità concatenativa** e **composizionalità funzionale**.

2.3.1 Composizionalità concatenativa

La composizionalità concatenativa in NLP (Natural Language Processing) si riferisce a un modo specifico di costruire **rappresentazioni di frasi** o testi combinando le rappresentazioni di parole individuali attraverso l'**operazione di concatenazione**. Questo approccio presuppone che il significato di una frase sia ottenuto concatenando le rappresentazioni delle parole che la compongono.

Esempio. Per comprendere meglio questo concetto, consideriamo un esempio semplificato. Supponiamo di avere le rappresentazioni vettoriali (word embeddings) per le parole "gatto," "mangia," e "pesce." Nell'approccio di composizionalità concatenativa, la rappresentazione vettoriale di una frase come "Il gatto mangia pesce" sarebbe ottenuta concatenando le rappresentazioni vettoriali di ciascuna parola in sequenza:

Rappresentazione della frase = Rappresentazione del "Il" Rappresentazione di "gatto" Rappresentazione di "mangia" Rappresentazione di "pesce" Rappresentazione della frase=Rappresentazione del "Il"Rappresentazione di "gatto"Rappresentazione di "mangia"Rappresentazione di "pesce"

Dove rappresenta l'operazione di concatenazione.
Questo approccio assume che il significato complessivo di una frase possa essere costruito in modo lineare combinando i significati delle parole individuali. Tuttavia, la realtà è spesso più complessa, poiché le relazioni sintattiche e semantiche tra le parole possono richiedere operazioni più sofisticate.

2.3.2 Composizionalità funzionale

La composizionalità funzionale in NLP (Natural Language Processing) si riferisce al concetto di costruire significati complessi attraverso la **composizione di significati più semplici**. In termini più semplici, significa che la rappresentazione di significati complessi è ottenuta **combinando** in modo sistemico le rappresentazioni di parole o frasi più piccole.

Nel contesto del NLP, questo approccio spesso coinvolge l'uso di modelli neurali, come le reti neurali ricorrenti (RNN) o i modelli basati su trasformatori, che sono in grado di catturare le relazioni semantiche tra le parole e le frasi. Questi modelli trattano le parole come vettori distribuiti, catturando le informazioni contestuali e semantiche.

Esempio. Un esempio pratico potrebbe essere la rappresentazione di una frase complessa come "il gatto nero è sul tappeto". La composizionalità funzionale implicherebbe la costruzione della rappresentazione di questa frase combinando le rappresentazioni delle singole parole ("gatto", "nero", "tappeto") in modo coerente per catturare il significato completo della frase.

Nella composizionalità funzionale, il modo di combinazione è una **funzione** Φ che dà un processo generale per produrre espressioni dati i suoi costituenti.

Le **local distributed representations** e le **one-hot encodings** sono il miglior modo per vedere come la *composizionalità funzionale* agisce sulle *rappresentazioni distribuite*. Ricordiamo che dato un insieme di simboli D, una rappresentazione distribuita locale mappa l'i-imo simbolo nell'i-imo vettore canonico di R^N, dove N è la cardinalità di D.Nella composizionalità funzionale, espressioni del tipo $s = w_1 w_w ... w_k$ sono ottenute con una funzione Φ applicata ai vettori $e_{w_1} ... e_{w_k}$, e questa funzione potrebbe anche essere la semplice somma, che ci dà come risultato un vettore **bag-of-word**, ma ci sono anche funzioni più complesse, come la *convoluzione circolare*; così però si perde la sequenza dei simboli. Oppure, il *prodotto scalare* tra due espressioni s_1, s_2 restituisce il numero di parole in comune. In generale, la funzione Φ è cruciale per determinare se i simboli saranno riconosciuti e la sequenza preservata.
Le *rappresentazioni distribuite* sono più ambiziose delle *rappresentazioni distribuite locali* perchè cercano si codificare simboli base di D in vettori di R^d dove d è molto minore di n, anche se così si alterano i simboli, data la assenza di link tra simboli e rappresentazioni. In generale, data la rappresentazione locale $\mathbf{e}_w$ di un simbolo w, l'encoder di una *rappresentazione distribuita* è una matrice $\mathbf{W}_{dxn}$ che trasforma e_w in $y_w = W_{dxn} e_w$. In una *rappresentazione distribuita* il contenuto informativo è **distribuito** tra **più unità**, ed allo stesso tempo ogni unità può contribuire alla rappresentazione di più elementi. I due vantaggi della rappresentazione distribuita rispetto alla rappresentazione distribuita locale sono che è più efficiente e non tratta ogni elemento come diverso allo stesso modo con ogni altro elemento. Il contro è che i simboli sono alterati e dunque difficili da interpretare.
Anche per le rappresentazioni dsitribuite è possibile definire **composizioni funzionali** per rappresenare espressioni. In generale, l'interpretabilità può essere a due livelli:

- **Symbol-level interpretability**: E' possibile riconoscere simboli discreti? Ovvero, a che livello la matrice di embedding W è invertibile?
- **Sequence-level Interpretability**: é possibile riconoscere i simboli e le loro relazioni in una sequenza di simboli? Ovvero, quanto sono concatenativi i modelli di composizione funzionale?

2.4 KERMIT, Zanzotto 2012

CAPITOLO 3

Analisi Sintattica

3.1 Analisi Sintattica (ai costituenti e alle dipendenze)

Come facciamo a creare un modello formale che funziona sempre nell'applicare le regole, ovvero nel costruire il linguaggio? La 'parola' è qualcosa che potrebbe essere composta da più lemmi (es: ha mangiato), e le individuiamo principalmente poiché divise da spazi (ma non sempre! es: daglielo).

> In informatica, il **parsing, analisi sintattica o parsificazione** è un processo che analizza un flusso continuo di dati in ingresso in modo da determinare la correttezza della sua struttura grazie ad una data grammatica formale. Un parser è un programma che esegue questo compito.

3.1.1 Teoria ai costituenti

In linguistica strutturale, l' analisi in costituenti è la teoria secondo cui la frase non è una semplice successione di termini connessi l'uno all'altro, ma è formata da una **combinazione di sintagmi**, a loro volta costituiti da unità più piccole, e così via sino a giungere agli elementi minimi indivisibili.

In parole povere, la teoria ai costituenti è l'idea per cui gruppi di parole si comportano come singole unità, dette costituenti. Gli elementi frasali (sequenze di parole che si comportano come unità) derivanti dalle categorie grammaticali, elementi ricorsivamente ricostruibili e non interropibili: i **sintagmi**, o **phrases**.

- NP, noun phrase: sintagmi retti da nomi (AGG+NOUN || ART+AGG+NOUN || NOUN+AGG || ART+NOUN+AGG)
- PP, prepositional phrase: sintagmi preposizionali (PREP+NP)
- VP, verbal phrase: sintagmi verbali
- A PP, adjective phrase: sintagmi aggettivali
- S, sentence: (?)

3.2 Constituency

Syntactic constituency è l'idea per cui si possono raggruppare parole che si comportano come singole unità: i costituenti. Parte dello sviluppo di una grammatica richiede di costruire un inventario di costituenti. Per esempio considero i **noun phrase**, sequenze di parole attorno ad un nome, queste appaiono sempre in ambienti sintattici simili, ad esempio prima di un verbo. Per caratterizzarli formalmente adottiamo le CFG, sistemi formali usati per modellare la struttura dei costituenti nel linguaggio naturale; la derivazione di una frase da una grammatica libera dal contesto può essere descritta da un albero di derivazioni, o albero sintattico, o albero di parsing, o ancora indicatore sintagmatico (phrase marker). Per riconoscere i sintagmi vengono utilizzati test di constituency, ad esempio abbiamo citato:

- spostamento: un sintagma è tale se può occupare diverse posizioni all'interno di un enunciato, le parole nella seq devono essere mosse congiuntamente;

  Osservazione 3.2.1 Lia partirà il prossimo mese. -> Il prossimo mese partirà Lia.

- sostituibilità: un sintagma è tale se può essere sostituito da pronome personale o preforme;

  Osservazione 3.2.2 Ale e Lia usciranno domani. -> Loro usciranno domani.

- non interrompibilità: "Un sintagma è tale se costituisce un'unità non interrompibile, cioè un costituente", se nell'inserire all'interno di una sequenza un morfema si ottiene una frase agrammaticale, si è spezzato un sintagma;

- scissione;
- enunciabilità;
- coordinabilità;
- ellissi;

3.3 Grammatica

Dato che vogliamo concentrarci sull'aspetto mentale del linguaggio, (sui princìpi che regolano il funzionamento della facoltà), facciamo affidamento alla **grammatica generativa**, una teoria del linguaggio con approccio derivante dalla teoria della dimostrazione, che studia ciò che le lingue naturali hanno in comune, piuttosto che ciò che le distingue. Ma cosa è una grammatica generativa? Un insieme di regole che "specificano" o "generano" in modo ricorsivo le formule ben formate di un linguaggio, (es: grammatica generativa dell'italiano). Le grammatiche riconoscono i linguaggi: possono stabilire se una stringa appartiene o meno ad un determinato linguaggio, decidere se una frase è grammaticale o meno.

Ci concentriamo sulle Grammatiche Context-Free, nonostante ciò è una limitazione rispetto al linguaggio naturale, in quanto alcune espressioni vengono tagliate fuori.

Osservazione 3.3.1 Ale e Maria mangiano **rispettivamente** un panino e una mela.

3.3.1 Context Free

Una grammatica libera dal contesto (o non contestuale, context-free o CFG) è una grammatica formale in cui ogni regola sintattica è espressa sotto forma di derivazione di un simbolo a sinistra a partire da uno o più simboli a destra. La grammatica formale, nella teoria dei linguaggi formali, è una struttura astratta che descrive un linguaggio formale in modo preciso, è cioè un sistema di regole che delineano matematicamente un insieme (di solito infinito) di sequenze finite di simboli (stringhe) appartenenti ad un alfabeto anch'esso finito. Ci semplifichiamo la vita ulteriormente perchè guardiamo alle Chomsky Normal Form, che riconoscono tutti i linguaggi context-free.

Chomsky Normal Form

In formal language theory, a context-free grammar, G, is said to be in Chomsky normal form (first described by Noam Chomsky) if all of its production rules are of the form:

$A \to BC$, or $A \to a$, or $S \to \varepsilon$,

where A, B, and C are non-terminal symbols, the letter a is a terminal symbol (a symbol that represents a constant value), S is the start symbol, and ε denotes the empty string. Also, neither B nor C may be the start symbol, and the third production rule can only appear if ε is in L(G), the language produced by the context-free grammar G.

Every grammar in Chomsky normal form is context-free, and conversely, every context-free grammar can be transformed into an equivalent one which is in Chomsky normal form and has a size no larger than the square of the original grammar's size.

Da CFG a CFN

Un linguaggio formale è definito come un insieme finito di possibili stringhe di parole. Si potrebbero avere due grammatiche equivalenti se generano lo stesso insieme di stringhe, e sono **fortemente equivalenti** se genenerano lo stesso insieme di stringhe e assegnano la stessa struttura frasale ad ogni frase; sono **debolmente equivalenti** se generano lo stesso insieme di stringhe ma senza la seconda condizione. E' dunque utile avere una **forma normale** per le grammatiche in cui le produzioni prendono una particolare forma (vedi **Chomsky Normal Form**).

3.3.2 Grammatica Universale

La grammatica universale, ovvero costruire una teoria della competenze, è l'obiettivo della grammatica generativa. La competenza, è costituita dall'insieme di conoscenze che permettono ad un parlante nativo di produrre messaggi verbali nella propria lingua.

3.4 Parsing ai costituenti

In informatica il parsing è un'attività che dato un input ne controlla la correttezza rispetto ad una certa grammatica formale. In altre parole, ne analizza la struttura.

Il nostro scopo è fare parsing di testi, che riduciamo a parsing su frasi, in quanto questo ci permette di estrarre informazioni che poi possiamo usare per analizzare la semantica e svolgere tantissimi task di NLP.

Il parsing ai costituenti si basa sull'idea che certi gruppi di parole possano comportarsi come unità singole.

- Token T: sono un concetto più astratto e generale rispetto alla definizione di parola come stringa contenuta fra due spazi; possiamo dire che un token sia un'unità significativa?
- NT: categorie lessicali: tags o part-of-speech, sono simboli che possono essere sostituiti con un T (nome, verbo, articolo,...), oppure i sintagmi, sono gruppi di parole che si comportano come una singola unità.

Osservazione 3.4.1 Esempio: Il lonfo non vaterca ne gruisce.

3.4.1 CYK: Cocke-Younger-Kasami

Algoritmo di parsing per grammatiche **context free**, che utilizza il bottom-up parsing e la programmazione dinamica. Il CKY restringe la forma della grammatica alla CNF, ma si limita a rappresentare tutti i possibili parsing della frase, senza scegliere un singolo parsing migliore.

Esempio su stringa visto a lezione 31/10/23

stringa: "baaba"

Regole Grammatica

- S -> AB
- S -> BC
- A -> BC
- A -> a
- B -> BB
- B -> CC
- B -> b
- C -> a

pseudo algoritmo Lungo la diagonale principale inseriamo i simboli che possono generare i terminali sottostanti, (i,i) := terminale[i];

(i,i+1) := combinazioni possibili di (i,i)x(i+1,i+1); ??? (i,i+2) := (i,i)x(i+1,i+2) and (i,i+1)x(i+2,i+1); ???

3.4.2 CYK probabilistico

Data una frase ed una grammatica, la quantità di alberi ai costituenti prodotta è esorbitante, il nostro scopo è ridurre il numero di alberi e trovare quello che meglio rappresenta la frase in quel determinato contesto.

$$Tmax = argmaxP\left(tree|S\right) = argmaxP\left(tree\right) = argmax\left(\prod\left(P\left(a->b|a\right)\right)\right)$$

Tuttavia questo ci impone forti assunzioni: tutti i pezzi dell'albero (le regole) sono indipendenti fra loro ed hanno la stessa probabilità indipendentemente dalla loro posizione (P(NP3 -> DT1 N2) == P(NP7 -> DT8 N9).

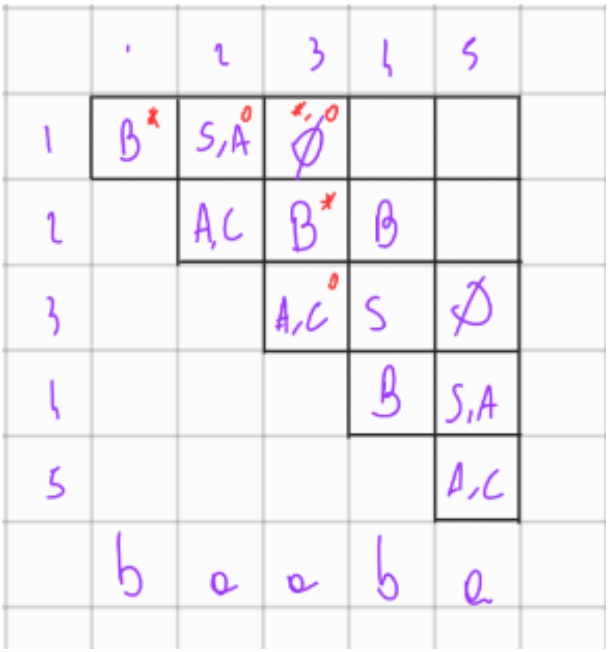


Figura 3.1. Esempio CYK visto a lezione, s='baaba'

(CYK) algorithm

input: $a_1 \dots a_n$.
$P_{n,n}$ is the matrix of sets of symbols initially $P_{n,n} = \varnothing$
For each i = 1 to n
 For each unit production $R_j \rightarrow a_i$, set $P_{1,j} = \{R_j\}$
 For each i = 2 to n
  For each j = 1 to n-i+1
   For each k = 1 to i-1
    For each production $R_A \rightarrow R_B\ R_C$
     If $R_B \in P_{j,k}$ and $R_C \in P_{j+k,i-k}$ then $P_{j,i} = P_{j,i} \cup \{R_A\}$
If $R_1 \in P[1,n]$
 Then string is member of language
 Else string is not member of language

Figura 3.2. codice algoritmo dalle slides

3.4.3 Treebanks

I cosiddetti **corpora** (sing. **corpus**) linguistici sono collezioni, per lo più di grandi dimensioni, di testi orali o scritti prodotti in contesti comunicativi reali (per es., registrazioni di discorsi o articoli di giornale), conservati in formato elettronico e spesso corredati di strumenti di consultazione informatici, organizzati per facilitare le analisi linguistiche. Un corpus in cui ogni frase è annotata con un albero di parsing è chiamato **treebank**; i treebanks sono creati facendo il prsing di ogni frase con un parsing fatto a mano. Il **Penn Treebank Annotation Scheme** è un progetto che include vari treebanks in varie lingue. Le frasi in un treebank formano implicitamente una grammatica della lingua: dalle frasi parsate, si possono estrarre le regole CFG.

3.5 Parsing alle dipendenze

L'analisi logica per come la conosciamo non è semplice, non dipende da posizioni (aggettivi prima e dopo il nome) o regole troppo inflessibili... ma anche la stessa tokenizzazione delle 'parole' può presentare problemi (perchè dividere per spazi? il cinese non divide le parole con spazi). Invece di cercare una relazione parentetica, guardiamo alle relazioni fra le parole -> costruzione delle dipendenze, un meccanismo con al di sotto una grammatica ai costituenti, che si basa su un classificatore che restituisce le probablità di esistenza di una determinata dipendenza fra due costituenti. Semplifichiamo le cose immaginando di poter tokenizzare le parole usando i transformer, il problema successivo è la loro classificazione. (es: DET/DETERMINER, parole che modificano nomi o NP, includono articoli, aggettivi pronominali, etc ...).

La struttura che possiamo realizzare guardando alle dipendenze, può essere un albero, se non consdieriamo archi incrociati e che ogni nodo abbia al massimo un nodo entrante, o più in generale un grafo, se guardiamo ad una versione enforced dove è ammesso tutto, ma non siamo più nel mondo context-free.

3.5.1 Universal Dependencies

Vorremmo che l'analisi sintattica fosse indipendente dalla lingua, ovvero trovare una modalità (semi) univoca per rappresentare le relazioni fra parole, che vadano sempre dal modificato al modificatore. Come già accennato identificare il concetto stesso di parola può essere ambiguo, magari anche in una stessa lingua, immaginiamoci fra lingue differenti.

3.5.2 Shift Reduced Parsing

Lo "shift-reduce parsing" è una tecnica di parsing utilizzata nell'analisi sintattica di un linguaggio. Questa tecnica è comunemente associata all'analisi sintattica ascendente (bottom-up) utilizzata nella costruzione di alberi sintattici o nella derivazione di grammatiche.
Si fa uso di uno **stack**, sul quale si costruisce il parsing, e di un **buffer** di token su cui eseguire il parsing, ossia la frase in linguaggio naturale. Inoltre, è necessario l'utilizzo di un **oracolo** che indichi quale operazione eseguire di volta in volta.
La frase viene analizzata da sinistra verso destra, portando di volta in volta gli elementi dal buffer allo stack. Ad ogni passaggio, si esaminano gli elementi in cima allo stack e l'oracolo indica quale operazione (**transizione**) applicare per costruire il parsing.
Le transizioni possibili sono le seguenti:

- *Shift* (Spostamento): Durante questa fase, il parser sposta un simbolo dal buffer alla pila degli elementi analizzati. In altre parole, il parser "prende" il simbolo successivo dall'input e lo aggiunge alla pila.
- *Reduce* (Riduzione): In questa fase, il parser cerca una regola nella grammatica la cui parte destra corrisponda ad una porzione della pila degli elementi analizzati. Se trova una corrispondenza, sostituisce quella porzione della pila con il lato sinistro della regola, riducendo in tal modo lo stack.

Questi due passaggi vengono ripetuti finché non si raggiunge uno stato accettante o si completa l'analisi dell'input. L'obiettivo è costruire un albero sintattico riducendo progressivamente la pila degli elementi analizzati fino a raggiungere uno stato finale. Il "shift-reduce parsing" è spesso implementato con l'ausilio di una tabella di parsing, che specifica le azioni da intraprendere (shift o reduce) in base allo stato corrente del parser e al simbolo corrente nell'input. Questa tecnica è ampiamente utilizzata negli analizzatori

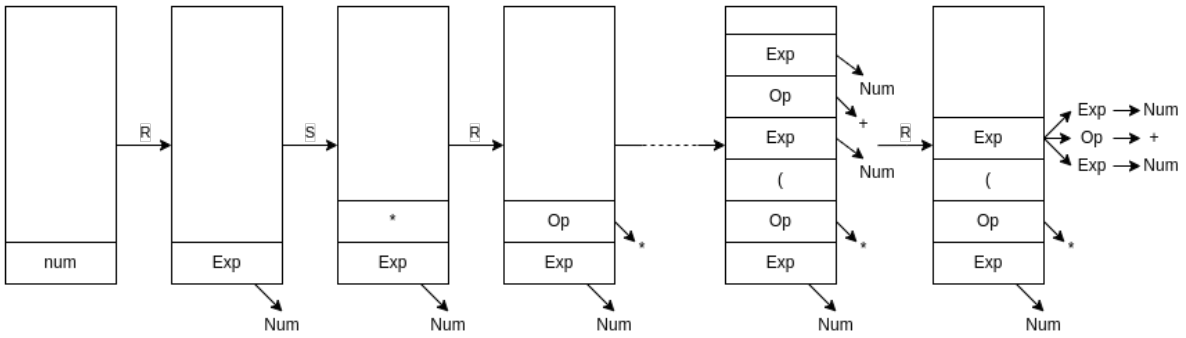


Figura 3.3. Alcuni passaggi dello shift-reduce parser applicato all'esempio

sintattici LR (left-to-right, rightmost derivation) come LR(0), LR(1), etc. In breve, lo "shift-reduce parsing" è un metodo efficiente per analizzare sintatticamente una stringa di input, costruendo un albero sintattico mediante uno schema di spostamenti e riduzioni.

Andiamo a scrivere una grammatica generale:

$Exp \to ExpOpExp$
$Exp \to -Exp$
$Exp \to (Exp)$
$Exp \to Num$
$Op \to +$
$Op \to -$
$Op \to *$
$Op \to /$

Allora lo Shift Reduce Parsing funziona alla stessa maniera. Se ho *num * (num + num)*, allora mette num, scandisce le regole e fa reduce, mettendo Exp sullo stack e collegandolo a Num. Non serve il concetto di left o right, perché sto costruendo un albero di interpretazione a costituenti. Questo era un parser precedente per analizzare dei linguaggi formali, dove formale significa la prima interpretazione è quella corretta, perché ci sarebbero altri alberi costruibili ma si predilige il primo, basta un algoritmo lineare e non cubico. Partendo da un corpus pieno di frasi, addestro su questo un classificatore che si occupa di fare le veci di una grammatica (ai costituenti), e suggerisce il prossimo step shift, reduce left o reduce right.

- Shift:
- Reduce Left:
- Reduce Right:

Osservazione 3.5.1 Esempio: Mario mangia la mela con il coltello.

3.6 Part Of Speech (POS) Tagging

Nella linguistica dei corpus, il **POS tagging** è il processo di contrassegnare una parola in un testo come corrispondente a una parte particolare del discorso, in base sia alla sua definizione che al suo contesto. Il PoS tagging consiste nel classificare per tag le parole, guardando al contesto, che aiuta a diminuirne l'ambiguità. La scelta stessa dell'insieme di parole deve essere informativa e duducibile dal contesto circostante, inoltre è desiderabile che la scelta avvenga 'il più localmente possibile' e che sia utile al livello successivo. parole: w1,w2,...,wn tag: t1,t2,...,tn Invece, la scelta dei tag è affidata ad un annotatore, un oracolo che conosce la sequenza di tag 'corretti', un sistema come insieme di triplette posizione, word, tag, solitamenti realizzati dai linguisti.

Come possiamo valutare la bontà di un **pos-tagger**? Dato che comunque lo stiamo vedendo come un modello di machine learning che si occupa del task di classificazione lo possiamo valutare con precision, recall, F-measure e accuracy.

3.6.1 Rule Based PoS Tagger

Modelli che applicano una serie di regole ed usano il contesto per assegrare i POS tags.

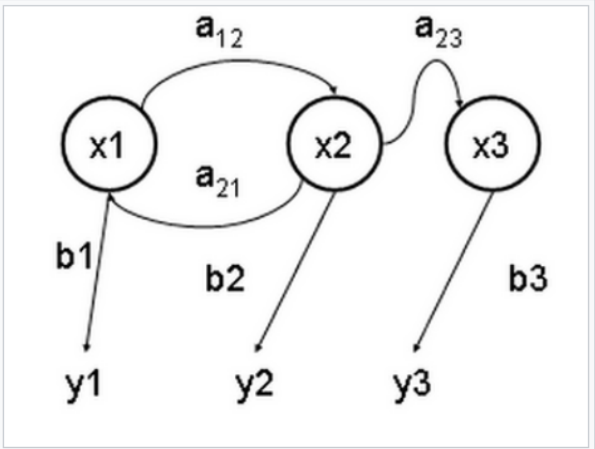
3.6.2 HMM: Hidden Markov Models

Sono usati per un approccio stocastico al POS tagging, dove le probabilità delle trasizioni sono la likelihood di una sequenza. Per definizione, Un modello di Markov nascosto (Hidden Markov Model - HMM) è una catena di Markov in cui gli stati non sono osservabili direttamente. Più precisamente:

- la catena ha un certo numero di stati;
- gli stati evolvono secondo una catena di Markov;
- ogni stato genera un evento con una certa distribuzione di probabilità che dipende solo dallo stato;
- l'evento è osservabile ma lo stato no.

Catturano incertezze e dipendenze temporali, vengono usati per derivare e spiegare le caratteristiche probabilistiche di un random process, ovvero quando il sistema o processo sottostante che genera le osservazioni è sconosciuto o 'hidden'. Nel caso del NLP vengono adottati per predirre osservazioni future o classificare sequenze.
Ci sono 3 **problemi canonici** connessi con gli HMM:

- Dati i parametri del modello, calcolare la probabilità di una sequenza particolare dell'uscita. Questo problema è risolto dall'algoritmo di forward.
- Dati i parametri del modello, trovare la sequenza più probabile che potrebbe generare una data sequenza dell'uscita. Questo problema è risolto dall'algoritmo di Viterbi (Andrea Viterbi).
- Data una sequenza dell'uscita o un insieme di tali sequenze, trovare l'insieme più probabile per il quale si possano dichiarare le probabilità dell'uscita e di transizione. Questo significa "addestrare" i parametri dell'HMM dato mediante il gruppo dei dati relativi alle sequenze. Questo problema è risolto dall'algoritmo di Baum-Welch.



Transizioni di stato in un modello di Markov nascosto (esempio)
x — stato nascosto
y — uscita osservabile
a — probabilità di transizione
b — probabilità di uscita

Lavoriamo sul tagging probabilisticamente:

$$Tmax = argmaxP\left(T|W\right) = argmaxP\left(W|T\right)P\left(T\right) = argmax\prod\left(P\left(wi|ti\right)P\left(ti|ti-1\right)\right.$$

abbiamo fatto varie assunzioni: i tag dipendenti solamente dallo stato precedente, le parole indipendenti fra loro e dipendenti solo dal tag, e altro. Il termine HIDDEN sta ad indicare che la parte da stimare markovianamente è il tag e non lo conosciamo. Tuttavia le sequenze di tag considerabili sono a prescindere tantissime, per evitare di considerarle tutte quante possiamo creare un grafo navigabile.

Montecarlo search

Beam Search

Viterbi Algorithm

3.7 Equivalenza fra parsing ai costituenti ed alle dipendenze

Nel caso del parsing alle dipendenze mi concentro sulle parole e come devo collegarle piuttosto che sulla struttura, che invece è il focus nel caso del parsing ai costituenti.

*** Non so il nome del paper o dei tizi che comunque hanno visto sono equivalenti, e si può passare da uno all'altro. ***

3.8 Concordanza tra parole

Estensione alle grammatiche context free, ma aumentiamo davvero il potere della loro rappresentazione? (no (?)).

Osservazione 3.8.1 Il vecchio porta la sbarra.

Consideriamo la frase appena riportata, per ogni parola andiamo ad estrarre numero e genere. Oltre al numero di elementi, c'è qualche particolarità? Ci siamo concentrati su due caratteristiche, potrebbe essere utile usare delle variabili (tipo struct) che le raccolgono tutte. Dobbiamo ragionare su come far uscire degli elementi interessanti e soprattutto come trattarli. Ad esempio, osservando la frase d'esempio notiamo che c'è qualcosa che dovrebbe legare DT a NN, ma questo qualcosa cosa è? Definiamo la variabile numero: [NUM]. Ci stiamo riferendo alle variabili matematiche, che in un sistema di equazioni devono avere lo stesso valore ovunque esse siano; diverse da quelle informatiche che sono spazi di memoria. Qui dobbiamo tornare al concetto di variabile matematica, deve avere lo stesso valore in tutta la regola. Vincolo: se riconosco un numero in un articolo, questo si propaga; questo concetto si chiama **unificazione** di una variabile. Se inizio a pensare a queste come variabili, vorremmo scrivere regole differenti, del tipo:

$$\begin{bmatrix} T:NP \\ G:x \\ N:y \end{bmatrix} \longrightarrow \begin{bmatrix} T:DT \\ G:x \\ N:y \end{bmatrix} \begin{bmatrix} T:NN \\ G:x \\ N:y \end{bmatrix}$$

Ma ci sono delle limitazioni: siamo passati a context-sensitive? Ci bastano gli algoritmi visti fin ora? Potremmo mettere molte altre caratteristiche. Il concetto che ci serve non è essere uguale, ma l'essere **compatibile**. Il verbo per esempio può avere un gran numero di vincoli. Abbiamo rinuniciato al concetto di uguaglianza, noi in CF dicevamo se la parte destra della regola è uguale è quella che osserviamo, nel parsing posso mettere la parte sinistra (vedi CYK), usando il concetto di uguaglianza; qui non posso usare questo concetto. Se abbiamo una feature con un nome, in tutte e due devo avere gli stessi valori o valori che possono subire questa trasformazione che stiamo generando, che non è di uguaglianza. Se le feature structure sono del tipo feature:valore, allora vorrei che la **feature structure** A sia compatibile con B (e non il contrario), ovvero mantengono lo stesso valore, le altre no. Se A ha una feature non compatibile in B, non va bene. Nell'unificazione vorrei far sia che

$$AUB = C$$

solo se A,B sono compatibili. In C ci devono stare tutti gli elementi di A e B solo se sono compatibili tra loro. Le info in più che non sono espresse da una delle due vengono mantenute. Questa operazione di unificazione dice che A deve *sussumere* C e B (si scrive A⊆C, B⊆C). L'unificazione viene fatta anche per le feature che hanno un valore composto (da una altra feature structure). Ora vediamo di strutturare

l'informazione. Ho magari una lista [T,GR,MORF,SEM] di queste informazioni. Tutte queste cose possono generare delle grammatiche molto informate, ad esempio che un verbo puo avere solamente soggetti umani, o soggetti animati, altri verbi magari possono averli inanimati. Da questo si è pensato di costruire grammatiche complicate che esprimono queste info nello stesso modo. Queste grammatiche sono state chiamate **Head-Phrase-Structure-Grammars (HPSG)**, studiate principalmente in Germania, perchè la lingua tedesca è piuttosto ricca in casi. Un'altra versione sono le **Lexical-Function-Grammars (LFG)**, ma con la stessa idea di avere simboli grammaticali strutturati che portino più informazione.

3.8.1 Features structure

Invece di realizzare una grammatica con una marea di regole per cercare di raccogliere le informazioni sulla concordanza delle parole (tipo DET SING FEM, DT SING MAS, etc...), sfruttiamo le variabili ???, ponendo sotto il concetto di variabile più informazioni/features (come una struct/object). Cambiamo la scrittura delle regole destrutturando il simbolo, passando dall'essere 'uguale' all'essere **compatibile**.

3.8.2 Sussunzione

: Il concetto di "sussunzione" (o "subsumption" in inglese) nell'ambito delle strutture di tratti nel contesto della linguistica e dell'elaborazione del linguaggio naturale (NLP) si riferisce alla relazione tra due strutture di tratti in cui una struttura di tratti è più generale o più ampia dell'altra. Questo concetto è spesso utilizzato per modellare la gerarchia di specificità o dettaglio tra diverse rappresentazioni linguistiche. Nel contesto delle strutture di tratti:

- **Struttura di Tratti più Generale**: Questa è la struttura di tratti che copre una gamma più ampia di casi. Ha attributi e valori che possono sussumere o includere quelli di una struttura di tratti più specifica.
- **Struttura di Tratti più Specifica**: Questa è la struttura di tratti che rappresenta un caso più specifico o particolare. Può avere attributi e valori specifici che non sono presenti nella struttura di tratti più generale.

Ad esempio, considera due strutture di tratti rappresentanti di animali:

- Struttura di Tratti Generale:
  - Attributo: "Tipo"
  - Valore: "Animale"
- Struttura di Tratti Specifica:
  - Attributo: "Tipo"
  - Valore: "Mammifero"
  - Altri attributi specifici ai mammiferi

Nel contesto di sussunzione, la struttura di tratti generale sarebbe "Animale" e la struttura di tratti specifica sarebbe "Mammifero". La struttura "Mammifero" sussume la struttura "Animale", poiché tutti i mammiferi sono anche animali. Questa relazione di sussunzione è utilizzata per modellare le relazioni gerarchiche e di inclusione tra concetti e categorie linguistiche nelle rappresentazioni strutturate del linguaggio naturale.

3.8.3 Unificazione

: Nell'ambito delle feature structure, l'unificazione è un processo mediante il quale due strutture di tratti possono essere combinate in una singola struttura di tratti, preservando le informazioni specifiche di entrambe.

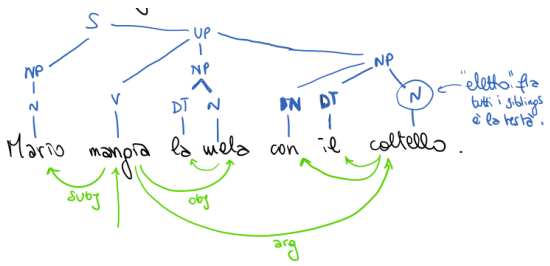


Figura 3.4. Equivalenza fra parsing alle dipendenze ed ai costituenti

Quando due strutture di tratti sono unificate, i loro attributi e valori sono combinati in una struttura più ampia. Questo processo consente di gestire casi in cui un'entità può avere più di un insieme di attributi o quando si desidera combinare informazioni da diverse fonti.

Ecco un esempio semplificato:

- Struttura di Tratti A:
  - Attributo: "Tipo"
  - Valore: "Mammifero"
- Struttura di Tratti B:
  - Attributo: "Colore"
  - Valore: "Bruno"

Quando unificate, queste due strutture potrebbero formare:

$$\begin{aligned} &StrutturadiTrattiUnificata: \\ &Attributo: "Tipo" \\ &Valore: "Mammifero" \\ &Attributo: "Colore" \\ &Valore: "Bruno" \end{aligned}$$

L'unificazione è spesso utilizzata nei linguaggi di programmazione basati sulla logica, come Prolog, per risolvere vincoli e rappresentare relazioni complesse. Nell'NLP, può essere impiegata per gestire la complessità delle rappresentazioni linguistiche, consentendo la combinazione di informazioni provenienti da diverse fonti o livelli di analisi senza perdita di dettagli.

Osservazione 3.8.2 Frase come "Aldo, Giovanni e Giacomo sono rispettivamente grande, piccolo e vecchio." restano comunque un problema che non riusciamo a risolvere con questa apparente espansione nella scrittura delle regole di una grammatica context-free.

3.9 Parser di Earley

Finora abbiamo visto solo CYK come algoritmo di parsing. Si deve scrivere ad esempio l'interpretazione per *il gatto mangia la mela con il verme*. Se la pensiamo con la rappresentazione arborea, avremo NP(il,gatto), V(mangia), NP(la mela), IN(con), NP(il verme), PP(con il verme), NP(la mela con il verme), VP(mangia la mela con il verme), S(tutto). Nella matrice del CYK vanno a

convivere diversi alberi tutti insieme, in una sola matrice. Accidentalmente qui è un albero binario, ma possiamo estendere tutto a un albero non binario. Andiamo a vedere questa innovazione: scrivere le interpretazioni relative alle frasi in maniera diversa, il che è correlato a quanto fatto con CYK. Se mettessimo dei **punti** tra una parola e l'altra che simboleggiano gli spazi,

$$.^1il.^2gatto.^3mangia.^4la.^5mela.^6con.^7il.^8verme.^9$$

e numerassimo questi punti (1,2,3,4,5,6,7,8,9) potremmo iniziare a scrivere le varie interpretazioni dei costituenti come essere degli archi tra questi punti, quindi il grafo correla questi punti e i **nodi** sono gli spazi tra le parole, mentre gli **archi** sono l'interpretazione di quella sequenza di parole; ad esempio, un arco tra 1 e 3 che chiamo NP, cioè NP(1,3), oppure NP(7,9), VP(3,9), S(1,9), e così via.

Se la penso sotto forma matriciale abbiamo lo stesso concetto. Se cancello ad esempio un arco, noto che stiamo rappresentando insieme **più alberi**. Iniziamo dalla regola $VP \rightarrow V, NP, PP$ che è un arco da 3 a 9, che ci dice quali costituenti servono per fare questa regola: stiamo compattando l'informazione così che in un solo grafo siano rappresentati più alberi.

Questa struttura dati si chiama **chart**, e se ci facciamo parsing si chiama **chart-parsing** (vedi *Earley Parser*). **chart[1]** indica tutte quelle strutture attive che arrivano nel punto 1, e così via. La genialata è rendere *attivo* questo grafo tramite il "puntino". In una regola sinistra destra, il puntino è messo allo scopo di dire fino a dove abbiamo utilizzato la regola, cioè quali parti abbiamo già analizzato. In uno stato della analisi potrei stare nella situazione in cui non sono arrivato al completamento della regola che volevo utilizzare, ma parzialmente ne ho analizzato una parte e si può usare su una porzione, ma non è completa (*regola **attiva***). Devo scrivere qualcosa che mi faccia andare avanti, ad esempio da $VP \rightarrow V - NP. - PP$, guardo la lista delle regole attivabili (devo fare la attività di **completamento** o **completer**).Per completare, posso dire che vorrei riempire un PP, devo ipotizzare che dopo mela inizi un PP; ora mi trovo accidentalmente che ho un PP (con il verme) completato, quindi posso spostare il puntino e spostarmi alla fine del PP e quindi ho completato anche questa.

Ora, dobbiamo iniziare da 0. Una grammatica che ci può servire è:

$$\begin{aligned} S &\rightarrow NP, VD \\ VP &\rightarrow V, NP | V, NP, PP \\ NP &\rightarrow DT, NN | NP, PP \\ PP &\rightarrow IN, NP \end{aligned}$$

Abbiamo a disposizione sempre:

$$.^1il.^2gatto.^3mangia.^4la.^5mela.^6con.^7il.^8verme.^9$$

e qualcuno ci ha detto che già per ogni parola abbiamo il relativo tag.
Dobbiamo fare **predizione** (o **predictor**). Parto dal punto 1, in modo top-down, ed ho

$$\begin{aligned} S &\rightarrow .NP, VP \\ NP &\rightarrow .DT, NN \\ NP &\rightarrow .NP, PP \end{aligned}$$

Ora, l'Earley Algorithm è top-down mentre il CYK è bottom-up. Cerchiamo di partire dai nodi, ovvero seguendo un approccio **bottom-up**.
Consideriamo sempre lo stesso esempio e grammatica. Conviene avere la grammatica in CNF, tengo un solo indice che va a essere spostato. Ci potrebbero essere delle regole **attive** o **attivabili**. (vedi *annotazione*, *chart parser*: le annotazioni sono state ampliate a contenere in ciascuno di questi archi una feature structure, cosi possiamo metterci piu caratteristiche usabili per attività varie, come definire che c'è un nome proprio, la locazione, etc...Il concetto di annotazione deriva dal parsing fatto con le annotazioni)

3.10 PN (Proper Noun), cosa è?

Abbiamo visto come i morfemi possono essere chiuse o aperti, e sono questi ultimi a portare contenuto. Esistono però degli elementi che sono necessariamente aperti e non sono catturabili con dei dizionari: questi ricadono negli elementi molto variabili, ma che hanno una regolarità. Il primo esempio è il nome che diamo alle persone, che non possiamo trovare in un dizionario. Ad esempio, *Vittorio Bianchi*, tipicamente scritto con le maiuscole; se consideriamo invece 'Vittoria'(o 'Bianchi'), non sappiamo se sono nomi oppure no, pensiamo al tedesco in cui le parole di contenuto sono sempre scritte con la maiuscola, o ad una trascrizione del parlato in cui non ci sono minuscolo e maiuscolo. Dunque in generale hanno una struttura intrinseca. In sostanza il nome proprio è un insieme di elementi che non possono essere catturati con dizionari, e vengono veicolati attraverso una struttura. Ad esempio, un nome potrebbe non essere compreso in un'altra lingua. Dove ritroviamo questo fenomeno? Indirizzi, email, indirizzi web.. che hanno tutti qualcosa di catturabile, perché hanno una struttura intrinseca, ma non possono essere scritti nella loro totalità in un dizionario.
Un altro esempio sono i numeri e qualunque cosa li contenga, ad esempio misure e soldi, elementi che non possono essere scritti in un dizionario; ad esempio '50 euro' e '150 euro' sono cifre diverse, e devono essere rappresentate in un altro modo (nelle PCFG non possiamo dare probabilità a ciascuna di queste). Si possono scrivere delle **espressioni regolari** per riconoscerli.
I nomi possono essere poi categorizzati (di persona, di luogo, etc). (vedi **NAME ENTITY**)

CAPITOLO 4

Analisi Semantica

4.1 Introduzione

Data la frase "Mario mangerà una mela, quante calorie ha ingurgitato?", tra le due frasi c'è distanza, come la colmiamo? Quale conoscenza ci servirebbe scritta attraverso dei testi? Ci serve di sapere le singole parole. 'Bere una mela' non si può fare, ChatGPT dice che dovrebbe essere trasformata in succo. Dovremmo dare a *mela* una quantificazione, al fine di arrivare a rispondere alla domanda sottostante, che richiede che noi sappiamo calcolare la quantità media di calorie in base al peso. Data la tipologia di mela dovremmo sapere la relazione tra i suoi attributi. Come la codifichiamo? La parola *ingurgitato* è correlata con *mangiato*, e se sì in che modo? Col significato. Quale è la relazione tra *ingurgitare* e *mangiare*? Mangiare ha come aspettativa una cosa solida, per esempio, come *tracannare* si aspetta liquidi. Ma se uno dice il verbo *vendere*, ha delle aspettative? Dei soldi scambiati, il compratore e il venditore, un bene; per *tracannare* invece una bevanda. Abbiamo fatto una supposizione importante, ovvero il **verbo è principe** e ci stiamo chiedendo come il verbo accoglie il resto. Tra due parafrasi ci potrebbe essere una rappresentazione in un'altra forma: potrei raccontare *vendere* come una composizione di eventi differenti. Vediamo ora se la stessa cosa potrebbe succedere con i nomi. Vediamo *animale*, *gatto*: questo introduce il concetto di *dizionario*, perchè posso descrivere il gatto in funzione del supertipo *animale*. In un dizionario ci sono due classi di parole: i nomi e i verbi (non ci sono aggettivi, avverbi,...), gli aggettivi modificano i nomi ma non i verbi. Tipicamente sembra che i verbi sono relazionati agli *eventi*, i nomi ai *partecipantiall'evento* e il nostro scopo è trovare eventi nei testi, e poi ragionare su questi eventi. Gli aggettivi vanno a modificare i nomi e gli avverbi sono qualità dei verbi e aggettivi, anche se in minore quantità. Tutto questo è quello che stavamo cercando di dire. Se vogliamo rappresentare il mondo, dobbiamo comprendere eventi e partecipanti, come si struttura un passaggio a questa rappresentazione ecc, a patto che tutto questo sia composizionale. Andremo a vedere:

- **Wordnet** e **Framenet**, portato in formato multilingua **Babelnet**, un accumulo di Wordnet tutti insieme
- **Semantica distribuzionale**: Ciò che c'era prima delle reti neurali, prima forma di vettorializzazione delle concettualizzazioni (Distirbutional Semantics)

Questo è il panorama, in più un altro elemento: Wordnet e Framenet ci danno una relazione *logica* dove applichiamo una **deduzione**, derivante da uno schema entity-relationship, poi ci renderemo conto che la logica non è sufficiente e occorre il **lambda calcolo**.

4.2 Uso della logica

Vediamo una frase semplice: "*Gianni ama Maria*": voglio costruire una **rappresentazione logica** sotto forma di **predicato**, dovremmo arrivare a *ama(Gianni, Maria)*, e poi vorremmo arrivare a formulazioni del tipo: *esiste x: ama(x, Maria)*. Come usiamo l'albero ai costituenti per costruire la rappresentazione semantica? Potremmo dire che quando *ama* sale si aspetta due argomenti; stiamo quindi risalendo l'albero dei costituenti. Dato che dobbiamo costruire delle rappresentazioni

logiche, vorremmo passare per la **logica**.

Parentesi di logica

"*Socrate è un uomo*". Uomo(Socrate). $\forall x$ Uomo(x) $\implies$ $Mortale\,(x)$ Il **modus ponens**, se A $\implies$ B, $\frac{A}{B}MP$. Noi vogliamo arrivare a *Mortale(S)*. Si dimostra eliminando il $\forall$ da $\forall x$ Uomo(x) $\implies$ $Mortale\,(x)$. Posso mettere una nuova formula o se è una di quelle precedenti o se deriva dalle due con una **regola di inferenza**. $Uomo\,(Socrate) \implies Mortale\,(Socrate)$ deriva dalle due col modus ponens. Ci serve un altro meccanismo per non eliminare simboli quali $\forall$ o l'esiste: il **lambda calcolo**.

4.3 Lambda calcolo

Il λ-calcolo fu inventato da Alonzo Church nel 1936 come fondamento del concetto di computabilità, e si bas su due concetti: **astrazione** e **applicazione**.

- L'**astrazione** è della forma

  $$\lambda x.t$$

  , ed è la definizione di una funzione che mappa x in t. Per esempio la scrittura

  $$\lambda x.x$$

  indica la funzione f(x)=x, ovvero l'identità.
- L'**applicazione** è della forma (t s), ed è l'applicazione della funzione t all'argomento s. Ad esempio, $(\lambda x.x\ 5)$ è l'applicazione della funzione identica al valore 5.

4.3.1 β-riduzione

Il processo di applicazione di una funzione ad un argomento si chiama β-riduzione:

1. si parte dall'espressione ($\lambda x.t$ N)
2. leviamo il prefisso iniziale λx.
3. sostituisci in t tutte le occorrenze di x con N
4. continua fino a che non possiamo fare ulteriori riduzioni

Esempio.

4.4 Rappresentazioni semantiche

Rappresentazione semantica: data una frase, restituirne una rappresentazione simbolica del significato. Ovvero,

$$Mariocorre \rightarrow corre\,(Mario)$$

Esempio

1. Espressione di partenza:
   - $(\lambda x.x\ 5)$
2. Leviamo $\lambda x.$
   - $x\ 5$
3. Sostituiamo "5" a x:
   - 5
4. Non possiamo fare altro quindi "5" è il risultato finale

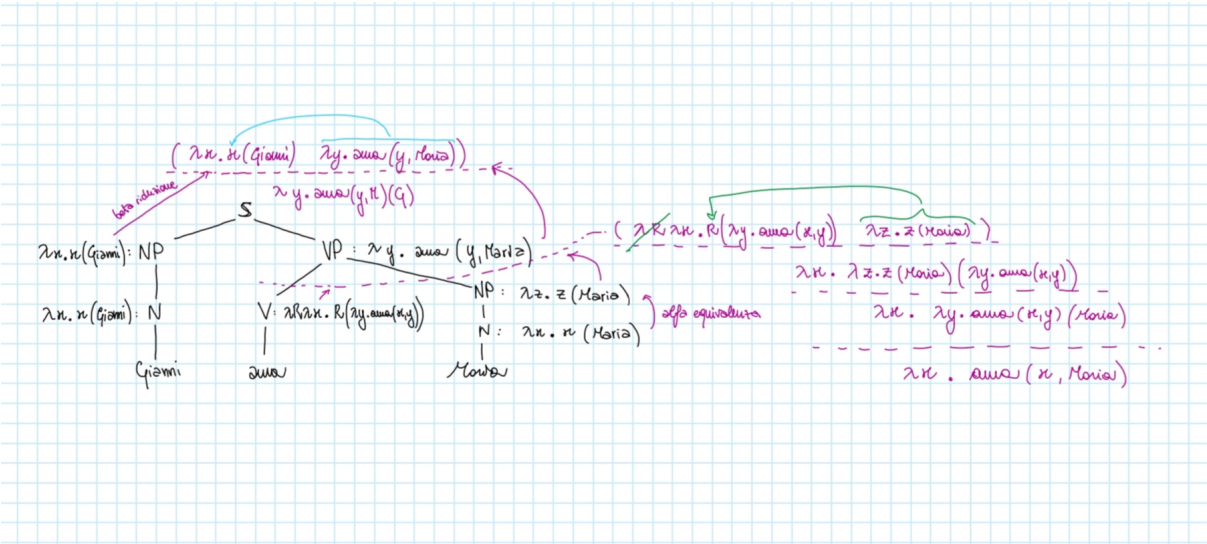


Figura 4.1. Enter Caption

$$GiovanniamaMaria \rightarrow ama\,(Giovanni, Maria)$$

$$OgniuomoamaMaria \rightarrow \forall x$$

4.5 Verso l'analisi semantica

L'analisi semantica nell'ambito del Natural Language Processing (NLP) è una branca che si occupa di comprendere il significato intrinseco del linguaggio naturale. In altre parole, si concentra sull'interpretazione e sulla comprensione del significato delle parole, delle frasi e dei testi in modo da consentire alle macchine di trattare il linguaggio umano in modo più simile alla comprensione umana.

Fino ad ora abbiamo cercato di seguire un approccio che portasse il linguaggio naturale verso un linguaggio logico, ovvero normare la realtà in base ad un linguaggio di arrivo (logica del primoprdine, il lambda calcolo); come vediamo nei linguaggi di programmazione, in cui si aggiunge alla logica la proceduralità.

Ad esempio, quando realizziamo delle basi di dati partiamo dall'analisi di un dominio, passiamo attraverso il diagramma ER, ed infine realizziamo la bd. (SQL non è logico, è ammisibile chiedersi 'A and NOT(A)' e si ottiene una risposta; vedi la legge della non contraddizione).

Data **f** una forma, la abbiamo rappresentata dapprima con una forma arborea, e solo successivamente siamo giunti alla **forma interpretata** nel linguaggio di arrivo **I(f)**. Cercheremo ora di andare direttamente alla forma interpretata, tramite una rappresentazione normata/normalizzata, non necessariamente logica. Ricordiamo il triangolo segnotico (di Peirce), in cui però stavolta non figura il referente. Ad esempio, nel caso del lambda calcolo, il processo prescinde dal linguaggio di arrivo, dato che si basa su formule, etc ...

4.5.1 Esperimento di Schank

L'esperimento teorizzato da Schank cercava di dimostrare che le persone ricordassero meglio alcune frasi o aspetti di esse in base alle "strutture" che venivano attivate. La ricerca si sposta dunque sull'individuazione di una base, tale da permettere di poter ricostruire i sensi in modo composizionale, esprimendo ogni parola in funzione di questa base. Quest'ultima viene reallizato riducendo al minimo i sensi, partendo dai verbi.

Osservazione 4.5.1 Trangugiare -> Mangiare -> Ingerire

4.5.2 Sostituzione salva significazione

Cerchiamo di partire dai verbi, l'obiettivo stavolta è raggrupparli in delle strutture che chiamiamo **synonym-set** (o synsets, o sensi): array di sinonimi, con cardinalità >= 1.

Una volta stabiliti i sensi, cerchiamo di trovare le relazioni fra di loro. Ci rendiamo conto molto velocemente, cercando su **WordNet** la parola 'dog', che in particolar modo con i nomi è molto facile avere delle strutture complicate e lunghe come risultato. Per scorrerle utilizziamo ipernimia (da un termine specifico risaliamo a termini più generali), iponimia (da un termine più generale scendiamo verso uno più specifico) o enteinment. Tuttavia questi simil grafi orientati sono una struttura fin troppo liguistica, per cui è difficile e complicato lavorarci, notiamo il problema dell'**ereditarietà multipla**, che ci porta ad avere dei cicli. Anche se con i verbi le strutture vengono più ridotte e meno complicate, non è detto che i problemi spariscano (vedi: derivationally related forms).

Lemmi diversi possono apparire in synsets diversi. Abbiamo gerarchie separate per nomi (tutti derivano da 'Entity', gerarchia molto ampia e ben specializzata), verbi(non c'è un unico grande verbo che fa da root, la gerarchia è più schiacciata) e aggettivi (sclta di design differente: si organizzano in coppie di aggetivi contrari, e poi si creano cluster attorno a ciascun eleento della coppia.)

Osservazione 4.5.2 " lemma.pos.num "

Word Sense Disambiguation

Come trovare il giusto synset per una parola?

4.6 Frame

I **frame** sono strutture dati impiegate nell'intelligenza artificiale per la rappresentazione della conoscenza. Questo concetto, introdotto da Marvin Minsky nel 1974 nell'articolo "A Framework for Representing Knowledge", consente di organizzare la conoscenza in sotto-strutture che rappresentano "situazioni stereotipate".
In termini più semplici, i **frame** sono le nostre aspettative in determinate situazioni. Nel campo dell'IA, vengono utilizzati per guidare il comportamento degli agenti intelligenti in modo predefi-

nito. Ad esempio, se un robot è programmato per agire come cameriere, deve seguire uno schema specifico quando prende gli ordini, e questo schema è definito tramite i frame.
Le parole esistono per soddisfare tre concetti chiave:

- **Informatività**: Serve a distinguere le cose, anche se talvolta può richiedere poche parole per farlo. Ad esempio, distinguere un quaderno da uno con anelli non richiede una nuova parola.
- **Economicità**: L'obiettivo è evitare un eccesso di parole per distinguere gli oggetti.
- **Naturalezza**: Le parole devono riflettere ciò che esiste nel mondo reale. Non si creano parole per concetti assurdi o inesistenti; una parola viene creata quando una situazione diventa stabile.

Questi principi contribuiscono a generare nuovi **concetti** e a creare un insieme di parole naturali, indispensabili per descrivere la realtà in modo accurato. In pratica, questo insieme ristretto di parole ci permette di creare in modo finito un insieme di frame basati su situazioni standardizzate. Ci sono due approcci distinti riguardo ai frame:

- **Approccio di Minsky**: I frame sono situazioni prototipali che guidano il comportamento standardizzato. Possono essere generati per qualsiasi situazione, e la sfida consiste nel crearne uno per ogni situazione possibile.
- **Approccio di Filmore**: Derivato dalle parole, raggruppa le lexical unit in frame, che rappresentano raggruppamenti di unità lessicali. Le frame contengono elementi necessari (core) e accessori, come tempo e modo.

4.6.1 Framenet

FrameNet è un progetto presso l'International Computer Science Institute a Berkeley, California, che produce una risorsa basata su frame semantici. Un frame semantico rappresenta situazioni, stati o eventi attraverso unità lessicali e ruoli semantici specifici per il frame. Il database di FrameNet contiene unità lessicali, frame semantici e frasi di esempio, consentendo la comprensione automatica del testo associando frame e ruoli semantici a ogni frase.

4.7 distributional semantics < NN

from sbd factorization, with loss function: guardo all'errore come distanza LTU to word embedding, differiscono per l'analisi, in questo caso ho anche un fattore di regolarizzazione.

skipgram: task prevedere la parola dato il contesto o viceversa ogni transformer viene addestrato con un pre-step, in cui avviene il word embedding.

4.8 Distributional Semantics

4.8.1 Intro

Partiamo da due parole: *gatto, cane*. Perchè sono più simili rispetto a *tavolo*? La tipologia di similitudine gerarchica la abbiamo vista con WordNet (si riconducono ad *animale*). In generale, ci

aspettiamo che abbiano **caratteristiche simili**: si muovono, mangiano, hanno le zampe, e molte altre caratteristiche in comune. Un altro modo dunque per avvicinare gli oggetti è guardare le caratteristiche, e mettendole in una gerarchia vediamo che scendono per il concetto di *ereditarietà*.
L'idea è quella di recuperare le caratteristiche dai **contesti** in cui trovo le parole.
Firth diceva che il significato delle parole lo determino dai contesti in cui si trovano, mentre Harris diceva che parole sono simili se si trovano in contesti simili. La concettualizzazione delle occorrenze in cui troviamo le parole stesse prende il nome di **contesto**.
Supponiamo di associare alla parola i contesti in cui la troviamo; per vedere se due parole sono simili dobbiamo trovare una **funzione di similitudine** tra i contesti.

Per definizione,

> La semantica distribuzionale comprende una serie di teorie e metodi di linguistica computazionale per lo studio della distribuzione semantica delle parole nel linguaggio naturale. Questi modelli derivano da una prospettiva empiristica e assumono che una distribuzione statistica dei termini sia preponderante nel delinearne il comportamento semantico.
> Questa teoria propone il paradigma per cui le parole sono distribuite in uno spazio nel quale sono, tra loro, ad una distanza proporzionale al loro grado di similarità. Quest'ultima segue l'ipotesi fondamentale della semantica distribuzionale (chiamata ipotesi distribuzionale) secondo la quale due parole sono tanto più simili semanticamente, quanto più tendono a comparire nello stesso contesto linguistico.

Data una collezione di documenti - corpus - trovo le occorrenze delle parole, ovvero il punto in cui si trova. Dal punto trovato, occorre definire cosa è un contesto, che dovrebbe essere qualcosa nell'intorno dell'occorrenza di una parola (come le parole all'intorno). Per ogni parola è auspicabile trovare numerosi contesti, e dunque è necessario unificarli. La grandezza del contesto fa sì che le cose possono essere molto diverse: il contesto potrebbe essere tutto un documento (è il massimo contesto ragionevole) - usato nell'ambito dell'information retrieval. Se usiamo il concetto di documento per generare il senso di una parola stiamo facendo una similitudine tra parole basata sui topic in cui queste appaiono (se una parola appare tante volte in documenti di sport sarà simile a un'altra parola nello stesso tipo di documenti).
Tornando al contesto di una parola, la grandezza del contesto si chiama **finestra**, e occorre dire se ci sono solo parole di contenuto (senza *stop-words*). Una finestra di dimensione 3 è la seguente:

$$w_{-3}w_{-2}w_{-1}\mathbf{gatto}w_1w_2w_3$$

Ora, vogliamo trovare una **funzione di similitudine** tra i contesti.
Potremmo pensare che più i contesti hanno parole in comune, più sono simili. Consideriamo come funzione

$$\frac{|G \cap C}{\sqrt{|G||C|}}$$

dove G e C solo unione di tutti i contesti di cane e gatto. Se consideriamo la rappresentazione *one-hot*, avremmo la sommatoria degli one-hot delle parole che appartengono ai contesti

$$\vec{G_1} = \sum_{w_i \in G_1} \vec{w_i} = [0,0,0,0,1,0,0,0,1,0,1]$$

Dunque,

$$\vec{G} = \sum_i \vec{G_i}$$

In G ci sono dunque le frequenze delle parole trovate nei contesti. Se facciamo il prodotto scalare diviso il prodotto delle norme 2:

$$\frac{< \vec{G}, \vec{C} >}{\sqrt{||G||||C||}}$$

abbiamo la **cosine similarity** tra G,C.

4.8.2 Ridurre la dimensione dello spazio

Come facciamo a ridurre questo spazio? Una soluzione era il **random indexing**. Altrimenti, consideriamo tutte le parole $w_1, w_2, ..., gatto, cane, ..., w_n$ che abbiamo trasformato in vettori. Siano queste le righe di una matrice, con colonne $w_1, ..., w_m$ (dunque, una matrice $M_{n\times m}$). Partendo dal sistema $A\vec{x} = \vec{y}$, consideriamo matrici U,V tali che $UU^T = I, VV^T = I$, allora $A = U\Sigma V^T \rightarrow A^{-1} = V\Sigma^{-1}U^T$, e la dimensione dipende da Σ (vedi singolar value decomposition (SVD), o vedi **Latent Semantic Index** (LSI) se ho come righe le parole e come colonne i documenti). (vedi Skipgram, prevedere la parola dato il contesto, o viceversa il contesto data la parola). La similitudine che abbiamo analizzato si chiama **cotopìa**, ovvero "stanno sullo stesso luogo".

4.9 Word2Vec

Word2vec è un insieme di modelli che sono utilizzati per produrre **word embedding**, il cui pacchetto fu originariamente creato in C da Tomas Mikolov, poi implementato anche in Python e Java. Word2vec è una semplice **rete neurale artificiale** a due strati progettata per elaborare il linguaggio naturale, l'algoritmo richiede in ingresso un corpus e restituisce un insieme di vettori che rappresentano la **distribuzione semantica** delle parole nel testo. Per ogni parola contenuta nel corpus, in modo univoco, viene costruito un vettore in modo da rappresentarla come un punto nello spazio multidimensionale creato. In questo spazio le parole saranno più vicine se riconosciute come semanticamente più simili. Per capire come Word2vec possa produrre word embedding è necessario comprendere le architetture **CBOW** e **Skip-Gram**.

4.9.1 CBOW e Skip-Gram

Word2vec può produrre word embedding utilizzando uno tra i seguenti due modelli di architettura: continuous bag-of-words (CBOW) e Skip-Gram[4]. Per addestrare i modelli è necessario disporre di un corpus di documenti testuali quanto più ampio possibile. Da questo corpus sarà possibile estrarre un vocabolario di parole distinte (token).

Osservazione 4.9.1 Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura

Ogni token distinto è quindi rappresentato con una codifica One-hot encoding. Questo approccio presenta però alcuni limiti poiché la dimensione del vettore V dipende dalla dimensione del vocabolario che potrebbe essere anche molto ampia. Questo enorme utilizzo di dati potrebbe manifestarsi nella curse of dimensionality. Inoltre, se venissero aggiunti/rimossi token dal word embedding, il modello dovrebbe essere riaddestrato da zero. L'ultimo grosso limite di questo approccio è dovuto al fatto che non viene risolto il problema della polisemia dei termini (meaning conflation deficiency). Nell'esempio di corpus sopra riportato sono presenti 13 token distinti, perciò la lunghezza del vettore V è necessariamente 13.

$$
\begin{aligned}
nel &[1,0,0,0,0,0,0,0,0,0,0,0,0,0] \\
mezzo &[0,1,0,0,0,0,0,0,0,0,0,0,0,0] \\
del &[0,0,1,0,0,0,0,0,0,0,0,0,0,0] \\
cammin &[0,0,0,1,0,0,0,0,0,0,0,0,0,0] \\
di &[0,0,0,0,1,0,0,0,0,0,0,0,0,0] \\
nostra &[0,0,0,0,0,1,0,0,0,0,0,0,0,0] \\
vita &[0,0,0,0,0,0,1,0,0,0,0,0,0,0] \\
mi &[0,0,0,0,0,0,0,1,0,0,0,0,0,0] \\
ritrovai &[0,0,0,0,0,0,0,0,1,0,0,0,0,0] \\
per &[0,0,0,0,0,0,0,0,0,1,0,0,0,0] \\
una &[0,0,0,0,0,0,0,0,0,0,1,0,0,0] \\
selva &[0,0,0,0,0,0,0,0,0,0,0,1,0,0] \\
oscura &[0,0,0,0,0,0,0,0,0,0,0,0,1,0] \\
\end{aligned}
$$

I due modelli esplorano il contesto dei token investigando gli altri token che sono prossimi a quest'ultimo. Qui si evidenzia la principale differenza tra le architetture CBOW e Skip-Gram.

Se la prima architettura mira a predire il token corrente (output) a partire da una finestra di parole di contesto (input); la seconda ha lo scopo di predire le parole di contesto (output) a partire dal token corrente (input).

Il parametro C definisce la dimensione della finestra di contesto: sono incluse nel contesto i C token immediatamente precedenti/successivi al token corrente.

CAPITOLO 5

Transformers

5.1 Transformers from scratch

5.1.1 Self-attention

L'operazione fondamentale per ogni architettura di transformer è l'operazione di **self-attention**. Self-attention è una operazione **sequence-to-sequence**: una sequenza di vettori è presa in input, e una sequenza di vettori và in output. Siano $x_1, x_2, ..., x_t$ i vettori di input e $y_1, y_2, ..., y_t$ quelli di output. Tutti i vettori hanno dimensione k.
Per produrre un vettore di output y_i, l'operazione di self-attention fa una *media pesata* su tutti i vettori di input:

$$y_i = \sum_j w_{ij} x_j$$

dove l'indice j scorre su tutta la sequenza, e i pesi sommano ad 1 su j. I pesi w_{ij} non sono parametri come nelle NN normali, ma sono *derivate* da una funzione di (x_i, x_j), come il prodotto scalare:

$$w_{ij}^{'} = x_i^T x_j$$

Il prodotto scalare però ci da valori tra $-\infty, +\infty$ e allora usiamo una softmax per schiacciare i valori tra (0,1):

$$w_{ij} = \frac{e^{w_{ij}^{'}}}{\sum_j e^{w_{ij}^{'}}}$$

Questa è l'unica operazione nell'intera architettura che propaga informazione tra i vettori, ogni altra operazione è applicata a ogni vettore nella sequenza di input senza interazione tra i vettori.

Per capire come funziona, prendiamo un esempio in cui abbiamo una sequenza di parole, *the cat walks on the street*. Per applicare la self-attention assegniamo ad ogni parola t un **embedding vector** v_t. Questo si chiama **embedding layer** nel sequence modeling. Quindi, la frase diventa una sequenza di vettori:

$$v_{the}, v_{cat}, v_{walks}, v_{on}, v_{the}, v_{street}$$

Se diamo questa sequenza in pasto al layer di self-attention, l'output sarà una nuova sequenza di vettori:

$$y_{the}, y_{cat}, y_{walks}, y_{on}, y_{the}, y_{street}$$

dove ad esempio y_{cat} è una somma pesata su tutti gli **embedding vectors** nella prima sequenza, pesati dal loro prodotto normalizzato con v_{cat}. Per capire però chi è v_t, vorremmo che il prodotto scalare esprimesse quanto due vettori sono relazionati nella sequenza di input in base al task di apprendimento. Ci sono due proprietà inusuali per una operazione sequence-to-sequence:

- **non ci sono parametr**i, perchè ciò che fà la self-attention è determinata dal meccanismo che crea la sequenza di input
- la self-attention vede il suo **input come un insieme**, e non una sequenza; se permutiamo la squenza di input, la sequenza di output sarà la stessa, anche se permutata.

5.1.2 Tricks

Il meccanismo di self-attention usa tre trick addizionali:

1. **Queries, keys e values**: ogni vettore di input x_i è usato in tre modi diversi nella operazione di self-attention:

   - **queries**: è comparato con ogni altro vettore per stabilire i pesi per il suo output y_i
   - **keys**: è comparato ad ogni altro vettore per stabilire i pesi per l'output del vettore y_j
   - **values**: è usato come parte della somma pesata per calcolare ogni vettore di output una volta stabiliti i pesi

   Ogni vettore di input gioca tutti e tre i ruoli nel meccanismo di self-attention di base. Per semplificarlo, deriviamo nuovi vettori per ogni ruolo, applicando una trasformazine lineare al vettore di input originale. In altre parole, aggiungiamo tre **matrici di pesi k*k**, ovvero W_q, W_k, W_v e calcoliamo tre trasformazioni lineari di ogni x_i per le tre diverse parti di self-attention.

   $$q_i = W_q x_i, k_i = W_k x_i, v_i = W_v x_i$$
   $$w_{ij}^{'} = q_i^T k_j$$
   $$w_{ij} = softmax\left(w_{ij}^{'}\right)$$
   $$y_i = \sum_j w_{ij} v_j$$

   Questo consente al layer di self-attention di avere dei parametri controllabili, e consente di modificare i vettori in entrata per essere compatibili con i ruoli che devono giocare.
   In sintesi, gli input sono i **values**, e un qualche meccanismo addestrabile assegna una **key** ad ogni value; poi ad ogni output, un qualche altro meccanismo assegna una **query**. I nomi derivano dalla struttura di uno store chiave-valore, dove però ci aspettiamo che un solo elemento nello store abbia la key che soddisfa la query, che viene ritornato quando la query viene eseguita. Il meccanismo di attention è dunque una versione più leggera, dove ogni *key* nello store soddisfa la query in qualche modo, e dunque sono tornate tutte, si fa la somma, vengono pesate in base a quanto la key soddisfi la query.

2. **Scaling del prodotto scalare** La funzione softmax può essere molto sensibile rispetto a valori di input grandi. Questo uccide i gradienti, e rallenta l'apprendimento, o ne causa lo stop. Dal momento che il valor medio del prodotto scalare cresce con la dimensione di embedding k, aiuta scalare il prodotto scalare per evitare che gli input della softmax function crescano troppo:

   $$w_{ij}^{'} = \frac{q_i^T k_j}{\sqrt{k}}$$

3. **Multi-head attention** Un fattore da considerare è che una parola può avere **diversi significati** in base a vicini diversi. Per esempio se considero la frase *"Mary gave roses to Susan"*, la parola *gave* ha relazioni diverse rispetto alle varie parti della frase (Mary esprime chi è che dà, Susan chi riceve). In una sola operazione di self-attention, questa informazione viene sommata insieme. Gli input x_{mary} e x_{susan} possono influenzare l'output y_{gave} con quantità diverse, ma non possono influenzarlo in modo diverso.
   Per dare più poterer di discriminazione, possiamo combinare **diversi meccanismi di self-attention** (con indice r) ognuno con diverse matrici trick W_q^r, W_k^r, W_v^r, e queste matrici si chiamano **attention heads**.
   Per un input x_i ogni attention head produce un diverso vettore di output y_i^r; li concateniamo, e li passiamo attraverso una trasformazione lineare per ridurre la dimensione di nuovo a k.

5.2 Costruire un Transformer

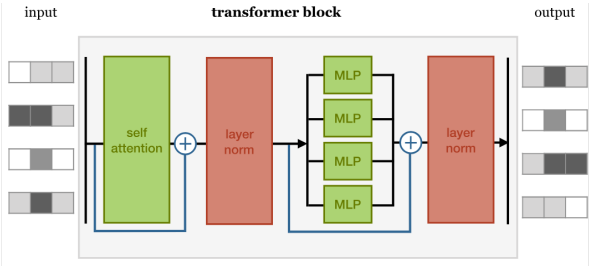
Un transformer non è solo un livello di self-attention, bensì è una architettura:

> Un transformer è una qualunque architettura progettata per processare un insieme di unità connesse - come i token in una sequenza, o i pixel in una immagine - dove l'unica interazione tra le unità è attraverso la self-attention

Vediamo ora un singolo **blocco** che possiamo ripetere.

5.2.1 Il transformer block

Ci sono molte varianti di come costruire un blocco, questa è una di esse: Quindi il blocco applica



in sequenza:

1. un livello di self-attention
2. un livello di normalizzazione
3. un livello feed-forward (un solo MLP applicato indipendentemente a ogni vettore)
4. un livello di normalizzazione

Le **residual connections** in blu vengono aggiunte ad entrambi prima della normalizzazione. L'ordine delle componenti non è fisso, basta combinare la self-attention con una feedforward locale, e aggiungerer normalizzazione e residual connections, che aiutano le DNN ad essere allenate più rapidamente e più accuratamente.

5.3 Intro

Data una sequenza a, c, a vorremmo che il transformer emettesse c. Prima si usavano le reti ricorrenti da termine a termine, e c'era il concetto di **attenzione**: questa dovrebbe farci prendere delle decisioni in funzione del contesto. Se chiamiamo

$$u_1, u_2, u_3$$
$$w_1 : a, w_2 : b, w_3 : a$$

allora si ha che l'uscita j-ima è una media pesata delle rappresentazioni degli elementi in ingresso, che dipende dalla posizione:

$$u_j = \sum_i \alpha_{ji} w_i$$

Questa media pesata viene fatta in funzione delle parole stesse, ovvero degli elementi di input. Nella sua versione più semplice, il termine α_{ij} è dato da

$$\alpha_{ij} = \left(w_j^T w_i\right)$$

Si perde l'ordine nella somma, nonostante siano distinguibili i singoli termini. Ciascuno di questi moduli, oltre alla parte di attenzione, ha una parte di **multi-layer perceptron** dopo.

Nella accezione della **key-query-value** abbiamo tre matrici Q, K, V, che vanno a moltiplicare nel seguente modo:

$$u_j = \sum \left((Qw_j)^T K w_i \right) V w_i$$

Si ha un **modulo di attenzione** che riceve l'input, che guarda il contesto

$$input \rightarrow ATTENTION$$

Poi, si somma l'uscita dell'attenzione e l'input normale; questo va in pasto a uno **strato feed-forward FFN**. L'output della FFN e l'uscita precedente vanno in pasto a una combinazione dei due in somma con normalizzazione.

Dotiamoci ora di uno strumento, che poi vedremo come usare. Se ora rappresentiamo in uno spazio a, b, c in cui sono tre vettori, tipicamente circa ortonormali. La domanda è: per fare in modo che nella posizione 3 esca fuori c, cosa mettiamo nelle altre matrici? Sapendo cosa gli vogliamo far codificare, come facciamo? Vogliamo fare uscire da b, a e da a, c. Allora scrivo prima

$$\vec{c} = A\vec{a}, A = ca^T$$

e poi

$$A = ca^T + ab^T$$

così se la moltiplico per b, data la ortonormalità, un prodotto fà 1 e uno fa 0 e rimane ciò che vogliamo.
Ora data una sequenza dobbiamo far uscire qualcosa. Oltre alla codifica della lettera c'è anche la **codifica della posizione**, un vettore indipendente dagli altri che sommo alla lettera stessa.

5.4 Symbolic vs Neuro

Questa distinzione racconta di un fatto trasversale, per cui esistono due modi di fare apprendimento, ovvero

- **Example-based (per esempi)**, detto anche *apprendimento **induttivo***: si presenta un insieme di esempi da cui bisogna derivare il modello, poi si applica il modello all'esempio successivo
- **Notion-based (per nozioni)**, detto anche *apprendimento **deduttivo***: si presenta il modello, e poi si chiede di applicarlo agli esempi

Tra induttivo e deduttivo c'è un *continuum*: quando dico una frase che sta a indicare delle informazioni necessarie a generare un modello, poi questo modello può essere applicato agli esempi, vedi **Rule-Based systems** in cui insegno alla macchina attraverso delle regole.
Vorremmo scrivere le regole in maniera simbolica e poi passarle nel neurale: questo è il tentativo di un Transformer.

La distinzione tra induttivo e deduttivo è caduta con i modelli **generativi**: questi lavorano su esempi di testo ma non hanno un target come i normali classificatori. Non avendo un target, è difficile categorizzarli come induttivi, perchè quando hanno appreso gli diamo delle istruzioni e quelli fanno un task specifico, che sembra essere deduttivo - ma non lo è. Tutto questo viene espresso come *neuro-symbolic models*, che mettono insieme le due cose, oppure *induttivo vs deduttivo*, anche se il concetto di transformers generativi si scontra con questo, oppure l' *in-context learning*. Dunque, tutto quello che si fà si trova nel mezzo di queste due cose.

Ora vediamo come passare da un simbolico a un rule-based.

5.5 Guidelines

Senza dare esempi, o dando un numero sempre minore di esempi insieme alle guidelines vorremmo una macchina che fa un task.
La prima ipotesi è costruire un **analizzatore di guidelines** che le leggesse e costruisse un modello operativo. Tipo, legge "se il tablet cade si rompe" avrebbe dovuto interpretarla e trasformarla in una regola logica per capire l'esempio "oggetto che si rompe". Questa idea si è provata a realizzare usando delle guidelines particolari. Si è preso un task RTE, e il tentativo era che dato l'esempio, si chiede a chatgpt di classificarlo. Poi, seguendo le guidelines, come classifica l'esempio, e con le guidelines lavora meglio che solo con gli esempi. Ci poteva però essere data-contamination, magari si sta solo triggerando il contesto. Allora si sono scritte altre guidelines, che spiegassero come fare il task in maniera differente, e il task viene performato peggio.

5.6

Supponiamo di avere l'idea che un task specifico, come il *question-answering*. Una domanda è "chi insegna ML?", e poi abbiamo lo **snippet** da cui estrarre una risposta che abbiamo trovato, "Russo Russo insegna ML" (task di classificazione domanda-smnippet ottenuti con **information retrieval system**). "Cosa insegna Russo Russo?", e lo snippet è sempre lo stesso "Russo Russo insegna ML". La struttura frasale nella seconda domanda è complicata, ho **chi** e **cosa** che hanno ruoli diversi. Voglio una regola che mi dica che se trovo "chi" e "insegna" allora ho il soggetto. Vorremmo correggere la macchina con pochi esempi, rappresentando sotto forma di albero sintattico. Una delle ipotesi usata è usare **dati sintetici**, uno ha una regola, ne deriva dati sintetici e fa apprendere alla macchina i dati sintetici. Devo dire quale sia la coppia domanda-risposta più probabile, e il terzo task è estrarre la risposta.
Ad oggi si fa **RAG Retrieval Augmented Generation**: la "retrieval-augmented generation" è un approccio che combina la ricerca di informazioni con la generazione di testo. Consente a un modello di linguaggio di attingere a fonti esterne per migliorare la precisione e la rilevanza del testo generato, ad esempio recuperando informazioni da una base di dati durante la generazione di risposte a domande.

Vogliamo mettere la regola dentro senza troppi giri, non è solo un modello di interpretazione del lavoro fatto da una NN, ma d'altra parte potrebbe essere un controllo della stessa. E'come se una feature che rappresenta un sottoalbero deve essere buttata o no.

Se ho questo albero

$$(s\,(NP, x)\,(VP\,(V, v)\,(NP, y)) \rightarrow \vec{v}$$

voglio codificarla in un vettore. Se ho

$$\vec{f} = v_1 + v_2 + v_3$$

per ognuno degli alberi che ci interessa prendiamo un vettore da una normale multivariata, con opportune caratteristiche; questa è una opzione, ma ci sarebbero un gran numero di alberi e non sappiamo quanti. Qual è lo spazio di tutti i sottoalberi che voglio rappresentare all'interno del mio modello, e quanto è grande tale spazio?

Vedi: *Distributed Tree Kernels, Zanzotto* e *Kermit:Zanzotto et al.* 5

Appendici

APPENDICE **A**

Machine learning

Qui verranno inserite tutte le brevi digressioni e cenni sul machine learning...

$BagOfWords -> WordsEmbeddings -> RNN -> LSTM -> Bi-directionalLSTM -> Transformers$

A.1 Tasks

NLP sta diventando l'applicazione del ML a problemi di elaborazione del linguaggio. Dato un linguaggio naturale, vogliamo trattarlo per svolgere possibili tasks:

- traduzione (corpora allineati);
- dialogo;
- QA;
- information extraction (da NL a Structured Language);

Questi task vengono definiti end-to-end (downstream tasks).

A.2 NN: Neural Networks

Le NN sono modelli per il machine learning costruiti usando principi dell'organizzazione neuronale: un gruppo di nodi interconnesso(neuroni) che si trasmettono segnali. Una rete neurale impara processando esempi contenenti un input ed un risultato noto/atteso per quell'input, costruendo associazioni probabilistiche e pesate, conservate nella struttura dati stessa della rete. Un neurone (artificiale) è una funziona matematica che riceve un input ne esegue una "somma" (con bias) e restituisce un output (es: sigmoid, step func, ...).

A.2.1 Long Short Term Memory: LSTM

Recurrent Neural Networks che permettono alle informazioni di persistere internamente fra i layers delle NN, e cercavano di rimediare al Vanishing Gradient Problem.

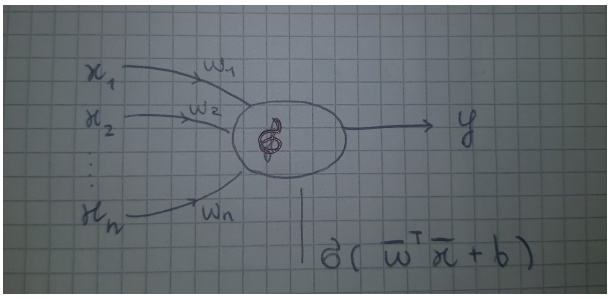


Figura A.1. Enter Caption

A.2.2 RNN: seq2seq

Esempio su un task di traduzione: consuma una parola per volta (vettore), prende in input la parola successiva, lo stato precedente e l'output vengono dati in pasto alla rete allo step successivo.

A.3 Transformers

Consideriamo l'input come una serie di vettori, applichiamo matrici, produciamo l'output intero.

APPENDICE B

Statistiche per il NLP

B.1 Bayesian Vs. Frequentist

Secondo i frequentisti: "Data are repeatable random samples from an underlying distribution, the probability of an event is its frequency (in the limit)". Invece per i Bayesiani "Data is what we observe from the sample, and are treated as distributions themselves". Nel secondo caso, è permesso includere informazioni precedenti e osservazioni soggettivi nei calcoli.

B.2 Inter-Annoters Agreement

Un modello è qualcosa che non esiste in natura costruito sulla realtà per fare delle predizioni; è un costrutto con cui noi esseri umani rappresentiamo il linguaggio, ad esempio un albero sintattico. Ma come ne giudichiamo la 'bontà'?

Consideriamo due o più persone (annotators) che si occupano di uno specifico task (linguistic) per testare il modello indipendentemente, se sono in accordo, allora questo potrebbe significare che il modello è simile alla rappresentazione del linguaggio che le persone hanno in mente. Come misuriamo il grado di agreement?

B.3 Misurare la qualità di un sistema

Dati due sistemi S1 ed S2, vogliamo valutare la somiglianza fra i due, se sono effettivamente lo stesso sistema o meno. I sistemi ricevono un determinato input C e formulano un risultato, un output simile implica che i due sistemi siano simili? Affichè le differenze rilevate siano statisticamente rilevanti, dobbiamo introdurre dei test significativi: **hypothesis tests**.

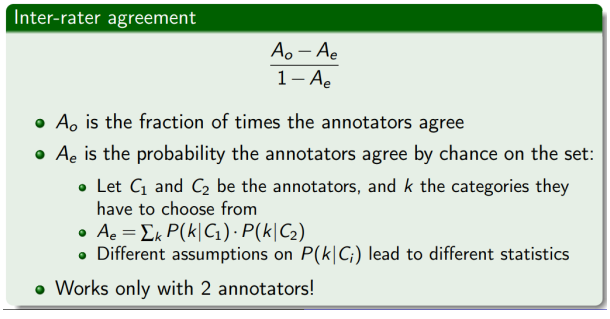


Figura B.1. Formula per Agreement (A)

Ipotesi Nulla: H0:= le due distribuzioni sono uguali. Ipotesi alternativa: Ha:= le due distribuzioni sono diverse.

Determinata l'ipotesi nulla cerchiamo di confutarla, impostando una soglia minima di verità (:= intervallo di confidenza). Le procedure che si occupano di restituire la probabilità con cui l'ipotesi nulla viene rifiutata sono significance tests, di seguito riportiamo gli esempi di test presentati a lezione, tutti applicati al caso .

B.3.1 Sign Test

Supponiamo i due sistemi siano modelli di machine learning che vengono valutati sulla metrica di accuracy, su un determinato test set, o meglio, raccogliamo le accuracy dei sistemi per k volte, su batches differenti. (vabbe avete capito, tipo k-fold) S1: + + + + - - + - + - S2: - - - - + + - + - + E conservo il segno della differenza fra le due accuracy. Anche in questo caso affinchè i valori siano statistiche significativi è preferibile applicare tecniche stile bagging andando a simulare più batches.

B.3.2 Wilcoxon Test

Non l'ho capito molto bene e non so neanche se è stato fatto quest'anno, non credo.

B.3.3 Yeh (2000)

Considero un terzo sistema O, oracolo. anche qua non ho molti appunti.

Conclusione

Siamo partiti da vedere il cervello umano da un punto di vista neuronale, ed abbiamo l'opportunità di guardare il comportamento di un cervello. C'è anche qualcuno che si occupa di replicare il cervello di organismi facendo le sinapsi nello stesso modo, allo scopo di vedere se gli spike si propagavano allo stesso modo. In questo momento abbiamo l'apparente modello del sostrato, dove naturalmente il nostro sostrato non è un feedforward ma è più complicato, non ha input o output.

Poi abbiamo avuto una rappresentazione a **blocchi funzionali**, in cui vediamo il cervello come un insieme di capacità diverse che comunicano tra loro. Questa rappresentazione funzionale è **simbolica**.