Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

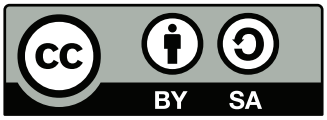
Corso di Laurea in Informatica Magistrale

Coreference Resolution basata su reti neurali deep

Relatore:
Chiar.mo Prof.
Fabio Tamburini

Presentata da:
Michele Corazza

Sessione 2
Anno Accademico 2016-17



Introduzione

L'utilizzo di reti neurali *deep* nell'ambito dell'elaborazione del linguaggio naturale sta conducendo negli ultimi anni a risultati significativi in task molto disparati, dalla *speech recognition* all'analisi semantica. La ragione di tali innovazioni risiede nelle capacità computazionali odierne, in grado di supportare l'utilizzo di reti neurali con molti livelli nascosti, dette appunto *deep*, e di strumenti innovativi quali le *recurrent neural network*, *convolutional neural network* e la possibilità di costruire *word embedding* tramite word2vec o strumenti analoghi.

Fra i *task* irrisolti nell'ambito delle reti neurali, è di particolare interesse lo studio della *coreference resolution*. In tale *task* l'obiettivo è quello di risolvere le coreferenze in un testo, ovvero associare menzioni che si riferiscono ad una stessa entità. Il fenomeno in esame risulta particolarmente interessante, in quanto comprende aspetti semantici e sintattici del linguaggio, che devono essere utilizzati per giungere a buoni risultati. Un ulteriore caratteristica della *coreference* è la relazione di tale fenomeno con il concetto di "contesto linguistico". È infatti dal contesto che circonda una menzione che è possibile intuire a quale entità esso si riferisca.

Le proposte per la risoluzione della *coreference* sono distinti in due principali categorie: vi sono approcci euristici, che sfruttano nozioni estrapolate dallo studio del fenomeno per cercare di risolvere tale problema, mentre gli approcci statistici sfruttano approcci basati su *machine learning*. Negli ultimi anni questi ultimi sono i più diffusi e, fra di essi, gli approcci basati su *deep learning* stanno ottenendo risultati promettenti.

Si presenta con questa tesi un *solver* per la coreference basato su reti neurali *deep*, che sfrutti reti *recurrent* per trattare il problema. La proposta si basa sulla supposizione che sia necessario introdurre delle componenti della rete che siano in grado di fornire una rappresentazione delle menzioni, in modo da poter utilizzare tali risultati per affrontare il problema della *coreference resolution*.

Nonostante non si sia riusciti completamente nell'intento di implementare un solver che abbia performance pari a quelle dello stato dell'arte, si è costruito un sistema relativamente ridotto rispetto a quelli proposti in letteratura, soprattutto per quanto riguarda i tempi di apprendimento della rete, che sono quantificabili in poche ore. Le risorse a nostra disposizione erano infatti molto limitate rispetto a quelle richieste dai sistemi con performance pari allo stato dell'arte, ragione per cui si sono dovuti operare diversi compromessi per ottenere qualche tipo di risultato, per quanto non ideale. L'utilizzo del sistema fornito nell'ambito di un meccanismo di *coreference resolution* più complesso potrebbe inoltre fornire migliori risultati, anche se tali sperimentazioni non sono per ora state realizzate.

L'implementazione del nostro sistema di *coreference resolution* costituisce in ogni caso un interessante caso di studio per le *recurrent neural network* applicate ad una problematica di complessa risoluzione e che richiede risorse computazionali ingenti.

Indice

Elenco delle figure

Elenco degli Algoritmi

Capitolo 1

Coreference Resolution

Nell'ambito dell'elaborazione del linguaggio naturale la risoluzione della catena delle coreferenze o dei riferimenti anaforici è da tempo un problema aperto. Prima di procedere ad una descrizione delle tecnologie e dei metodi risolutivi utilizzati in letteratura e nel progetto di tesi che si è sviluppato, risulta dunque necessaria una descrizione di ciò che si intende per *coreference resolution* ed *anaphora resolution*. La maggior parte degli esempi e delle informazioni presenti in questo capitolo sono stati adattati da "*Anaphora Resolution*" [48].

1.1 Riferimenti anaforici

Sebbene i termini *anaphora resolution* e *coreference resolution* vengano talvolta utilizzati in letteratura come sinonimi, dal punto di vista linguistico si tratta di due fenomeni distinti. Se l'*anaphora resolution* è legata alla presenza del fenomeno linguistico dell'anafora, la coreference riguarda un concetto più esteso, che comprende coreferenze di tipo anaforico e non. Per meglio comprendere il problema oggetto di questa tesi è indispensabile una descrizione dell'anafora, un fenomeno linguistico che presenta caratteristiche differenti a seconda della lingua in cui viene utilizzata, ma che risulta essere presente nelle lingue più comuni correntemente in uso.

Il concetto di anafora è legato indissolubilmente al contesto linguistico. Si parla di termine anaforico se vi è un elemento lessicale la cui semantica dipende da altri termini presenti nel testo che lo precede. Uno degli esempi più comuni di anafora riguarda l'utilizzo di pronomi che si riferiscono a entità già descritte nel testo.

> **Esempio 1:**
> Durante la passeggiata Fabio incontrò Mario. Lo accompagnò in stazione perché doveva prendere un treno.

Se esaminiamo la seconda frase in isolamento, appare evidente come il termine "lo" non contenga la definizione di chi sia stato accompagnato in stazione. Essa è infatti parte della frase precedente. Dal contesto linguistico possiamo dunque inferire che "lo" si riferisca a Mario e non, ad esempio, a Fabio.

Se in questo caso l'anafora è stata volutamente esplicitata da un pronome, nella lingua italiana esiste la possibilità di un riferimento anaforico implicito, nel quale viene omesso il pronome.

> **Esempio 2:**
> Mario aveva solo 10 minuti di tempo per arrivare in stazione. Non ebbe neppure il tempo di salutare gli amici.

Nelle frasi precedenti il riferimento a Mario come soggetto della seconda frase non viene esplicitato: se dal punto di vista della correttezza sintattica la presenza del pronome "egli" consentirebbe di esplicitare l'anafora, nell'italiano tale pronome può essere omesso. Si parla in questo caso di *zero pronoun anaphora*, quando cioè una forma foneticamente nulla si riferisce ad una entità reale.

Se nell'italiano (ma anche, ad esempio, nella lingua cinese [66]) tale fenomeno linguistico è comune, esso non è presente in tutte le lingue. In inglese, ad esempio, i pronomi devono essere sempre esplicitati. Pertanto non è possibile la presenza di *zero pronoun anaphora*. Vi sono però altre tipologie di anafora, nelle quali non viene utilizzato un pronome, bensì una frase nomi-

nale che, se estrapolata dal contesto, non codifica per intero l'entità a cui si riferisce.

> **Esempio 3:**
> Durante la passeggiata, Fabio intravide fra i pini marittimi una vecchia dimora signorile, da tempo in stato di abbandono. L'edificio era coperto da una fitta vegetazione, come se facesse ormai parte del paesaggio naturale.

Nella frase precedente è evidente come "edificio" non codifichi interamente l'entità a cui si riferisce. Se si considera solo la seconda frase, "edificio" risulta essere un termine generico, mentre, tramite il contesto presente nella prima frase, è ovvio che ci si riferisca alla dimora signorile abbandonata.

È importante evidenziare come le anafore introducano in maniera inevitabile ambiguità, che possono essere risolte solo attraverso il contesto linguistico che le circonda.

> **Esempio 4:**
> Alberto vide correre verso di lui il labrador. Il suo pelo, solitamente bianco, era coperto di fango a causa della pioggia del giorno precedente.

In questo caso il termine "suo" potrebbe, dal punto di vista della correttezza sintattica, riferirsi sia ad Alberto, sia al cane. Dal contesto, tuttavia, è evidente che non possa che essere il cane ad avere il pelo coperto di fango.

La tipologia di anafora fin qui presentata riguarda nominali. Esistono tuttavia altre tipologie di espressioni la cui interpretazione è dipendente dal contesto. Fra gli altri individuiamo:

- **Pro-verbs**: Andrew likes the same sports that I do.
- **Gapping**: Andrew drives cars, Robin _ bikes.

Nel caso del pro-verb "*do*", il significato completo della frase dipende dal verbo "*likes*", che è nella porzione precedente del discorso. Nel caso di *gapping*, invece, viene omesso il verbo, che viene estrapolato dal contesto. Un ulteriore esempio, ancora più estremo, riguarda espressioni verbali complete il cui vi è una componente *context-dependent*. Ad esempio, nella frase:

Esempio 5:
Carlos arrivò stanco a casa. Entrò in bagno e si fece una doccia.

Nell'esempio precedente, non è necessario introdurre, ad esempio, l'avverbio "successivamente" prima della seconda frase. Infatti è evidente dal contesto come l'entrata in bagno di Carlos avvenga successivamente all'arrivo a casa.

Sebbene si siano forniti esempi di altre tipologie di espressioni di tipo anaforico, tali tipologie risultano essere di minore interesse in ambito della linguistica computazionale. Essa infatti si concentra, nella maggior parte dei casi, nella risoluzione di anafore di tipo nominale. Ulteriori complicazioni derivano dal fatto che la esatta definizione di anafora, anche considerando solo i casi nominali, dipende dalla teoria linguistica a cui si aderisce e talvolta addirittura dalla valutazione personale di chi classifica le anafore in un testo.

Una volta descritto in breve il fenomeno dell'anafora, è utile introdurre alcuni termini linguistici specifici, che riguardano i fenomeni dell'anafora e della *coreference*:

1. **Referring expression**: si tratta dell'espressione che si riferisce ad un'entità definita in un'altra espressione;

2. **Anchor**: l'entità a cui si riferisce la *referring expression*;

3. **Antecedent**: un anchor nella quale la relazione con la *referring expression* è di identità.

L'intero scopo di un risolutore automatico di anafore è quindi quello di stabilire quali fra le espressioni siano *referring* ed individuare l'*anchor* ad esse legata.

Sebbene si tratti di un fenomeno presente in molte lingue, è evidente come la risoluzione dell'anafora non possa prescindere dalle caratteristiche intrinseche della lingua stessa. Se ad esempio in inglese esiste il genere neutro, in italiano esso non è parte della lingua. Si è inoltre visto come la presenza di *zero-pronouns* renda necessaria un'individuazione di tale fenomeno linguistico per risolvere l'anafora nelle lingue in cui esso è presente.

1.2 Corefence

Ci siamo finora concentrati sulla definizione di anafora e su alcuni termini fondamentali per la comprensione del problema della coreference ed *anaphora resolution*. Come si è già evidenziato, anafora e *coreference* sono due fenomeni linguistici affini ma distinti. In particolare, se nel caso dell'anafora la comprensione della *referring expression* dipende in parte o interamente dal contesto, nella *coreference* non è sempre questo il caso.

> **Esempio 6:**
> Gentiloni ha partecipato al forum dedicato alla *One Belt initiative* in Cina. [...] Il premier italiano ha espresso l'interesse del bel paese per i progetti di sviluppo proposti dalla potenza asiatica.

Rispetto ad un'espressione anaforica, è evidente come le due espressioni "Gentiloni" e "Il premier italiano" esprimano entrambe in maniera completa l'entità a cui si riferiscono. Si parla dunque di *coreference* quando due espressioni differenti si riferiscono ad una stessa entità, anche nel caso in cui non vi sia un riferimento anaforico. Rispetto alla risoluzione dell'anafora si tratta dunque di un fenomeno più ampio.

Abbiamo visto come le anafore introducano inevitabilmente ambiguità. Oltre a tali problematiche, per un risolutore automatico risulta complessa la distinzione di espressioni identiche o simili che però appartengono a catene di anafore coreferenze distinte.

> **Esempio 7:**
> Clinton$_A$ was a president of the US$_A$ between 1993 and 2001. He$_A$ then had to resign [...] [Hillary] Clinton$_B$ was a presidential candidate$_B$ in 2009 and 2016. She$_B$ won the democratic primary.[...] Donald Trump$_C$ won the election and became the 45th US president$_C$.

È evidente come vi siano due tipi di possibili errori nel testo precedente: da una parte il cognome Clinton si riferisce a due persone diverse. Dall'altra, il ruolo di presidente degli stati uniti è un ruolo elettivo, che quindi si riferisce a persone diverse. Un possibile errore di un risolutore automatico condurrebbe

dunque a creare, ad esempio, un'unica catena di coreferenza che comprenda le due catene A e B evidenziate nell'esempio precedente, oppure addirittura a fondere le tre entità A,B e C. In entrambi i casi avremmo un pronome maschile (*he*) che è coreferente ad un pronome femminile (*she*), una situazione che da luogo a evidenti incompatibilità.

Un'ulteriore situazione problematica riguarda la cosiddetta *non coreference detection*, ovvero la discriminazione dei nominali che sono referring expression. Fra di essi individuiamo anche le istanze di "*it*" nella lingua inglese, utilizzato nella sua forma impersonale:

> **Esempio 8:**
> It's a beautiful day.

Tali espressioni devono essere riconosciute e scartate per quanto possibile, in maniera da limitare le situazioni nelle quali essi vengono assegnati a catene di *coreference*.

1.3 Vincoli e preferenze

Dal punto di vista linguistico, la ricerca sulla *coreference* si è inizialmente focalizzata sull'individuazione di vincoli di vario tipo, che potessero fornire informazioni sul fenomeno della *coreference*. I vincoli individuati sono stati categorizzati come vincoli morfologici, sintattici e semantici.

Fra i vincoli morfologici troviamo, ad esempio, i vincoli di genere, che impediscono la coreferenza fra entità di genere diverso. È questo il caso di *he* e *she* nell'esempio precedente. Tali vincoli vengono individuati da studi psicologici come discriminanti che vengono applicati molto presto dal cervello umano per disambiguare espressioni anaforiche o coreferenti [5]. Tali vincoli introducono inoltre delle sostanziali differenze fra lingue con genere semantico (come l'inglese), nelle quali il genere sintattico è sempre causato da un genere dell'entità reale a cui ci si riferisce, e lingue con genere sintattico (come l'italiano e lo spagnolo), nelle quali anche i termini neutri hanno un sesso arbitrariamente stabilito dalla lingua stessa. Sebbene si tratti di un

vincolo con potenzialità di discriminazione molto ampie, esse vengono in parte ridotte da errori nell'utilizzo dei pronomi e da difficoltà dei sistemi automatici nel riconoscere il sesso di nomi propri non comuni o il cui genere è ambiguo.

Un altro tipo di vincolo morfologico è quello legato all'accordo di numero, legato all'impossibilità di avere coreferenza fra un nome singolare ed un nome plurale. Alcune difficoltà vengono però evidenziate nei sistemi automatici che sfruttano informazioni sul numero. È infatti difficile per un sistema automatico distinguere nomi singolari che sono però nomi collettivi, che quindi possono essere coreferenti con nomi plurali.

Esempio 9:
Il governo italiano si riunisce oggi in seduta straordinaria. Il presidente del consiglio e i ministri hanno discusso della crisi migratoria in atto.

Tale problematica dipende indubbiamente dalla definizione di coreferenza utilizzata, nonché dal contesto. Se intendiamo il governo italiano come un'istituzione (si pensi, ad esempio, al contesto di articoli della costituzione italiana) esso non risulta essere coreferente con gli individui che lo formano, mentre se si vuole esprimere il governo come gruppo di individui, esso è indubbiamente costituito dal presidente del consiglio e dai ministri.

Oltre ai suddetti vincoli morfologici, esistono anche vincoli sintattici, che dipendono da regole ben precise e che sono state oggetto di studi di una branca della linguistica chiamata *binding theory*. L'obiettivo di tale branca è la creazione di un sistema di regole formali per cui nell'esempio seguente *him* non può riferirsi a John, mentre in *himself* deve riferirsi a John.

Esempio 10:
John likes him.
John likes himself.

Pur senza descrivere complesse teorie linguistiche, che non sono parte dello scopo della tesi, è utile notare come negli anni siano state proposte diverse variazioni alla relazione *command* proposta da Langacker [35], che, unita a regole ben precise, è in grado di fornire vincoli sintattici in modo rigoroso.

Discutiamo infine il concetto di vincoli semantici, che sono legati al senso generale della frase. Nella frase:

> **Esempio 11:**
> Alfredo non ha un cane. [...] Esso è nella sua cuccia.

Dal punto di vista morfologico e sintattico, non vi è nessun problema nell'ipotizzare che "esso" si riferisca a "un cane". Tuttavia è stato stabilito in precedenza come Alfredo non possieda un cane. Questo tipo di situazione non può essere individuata se non utilizzando il significato complessivo della frase, ovvero la sua semantica. È evidente come, rispetto a vincoli sintattici e morfologici, che possono essere codificati in maniera formale ed utilizzati, sebbene non in modo perfetto, da sistemi automatici, i vincoli semantici risultino impossibili da definire in modo formale e siano quindi molto difficili da applicare in pratica.

Sebbene si siano discusse le principali categorie di vincolo applicabili alla *coreference*, esse non consentono di eliminare del tutto la ambiguità. Si sono per questo individuati dei criteri di preferenza, in grado di fornire dei ulteriori strumenti per la scelta di eventuali relazioni di coreferenza ed ulteriormente ridurre i casi ambigui.

Fra tali criteri di preferenza,vi è la plausibilità basata sul senso comune. Si considerino i seguenti esempi:

> **Esempio 12:**
> Gli studenti si riunirono con i professori perché essi avevano intenzione di occupare la scuola.
> Gli studenti si riunirono con i professori perché essi volevano indire uno sciopero.

Sebbene le due frasi siano identiche fino ad un certo punto, è sufficiente cambiare il verbo per cambiare una preferenza dagli studenti al corpo docenti. In questo caso sono le informazioni sul senso comune a fornire informazioni fondamentali per risolvere la *coreference*: gli studenti, infatti, non possono indire scioperi, mentre i professori tendenzialmente non occupano le scuole.

Oltre al tipo di preferenza basata su senso comune, sono stati studiati in letteratura varie casistiche nelle quali il senso comune è fondamentale per risolvere le *coreference* [12, 29, 61].

Vi sono poi dei criteri di preferenza sintattica. Da statistiche sui corpus [1], si evince come, nei corpora in lingua inglese, circa il 60-70% dei pronomi abbia il ruolo di soggetto nel discorso. Fra questi, il 70% hanno un antecedente che è anch'esso soggetto [48]. Questo fenomeno, denominato "assegnamento del soggetto", è stato oggetto di studi approfonditi nell'ambito della psicolinguistica [10, 18]. Un fenomeno analogo è stato osservato per quanto riguarda i pronomi in posizione di oggetto, che tendono a riferirsi ad altre espressioni che hanno il ruolo di oggetto.

Nella selezione degli antecedenti è fondamentale considerare come criterio di preferenza anche la *salience*, la cui forma più semplice, detta *recency*, si basa sulla distanza dell'*anchor* dalla *referring expression*. Quest'ultima tende infatti a riferirsi ad *anchor* che sono nelle frasi ad essa immediatamente precedente. Questo fenomeno ha dato vita in linguistica a studi approfonditi sulla cosiddetta *centering theory* [25], che studia come ciascuna nuova espressione influisca sul focus locale del discorso e quindi sulla probabile posizione di un antecedente di *referring expression* della frase successiva. Le informazioni sulla salience non consentono però di semplificare in modo estremo la risoluzione di *coreference*: selezionando sempre il primo candidato in accordo di genere e numero con la *referring expression* si ottengono risultati mediocri (60% di accuratezza [56]). Oltretutto non vi è un limite di distanza in termini di frasi fra la *referring expression* e l'antecedente. È tuttavia utile considerare la distanza fra le due espressioni come un parametro importante nella selezione dei candidati durante la *coreference resolution*.

[1] Si intende per corpus un insieme di documenti in cui sono stati annotati rispetto al fenomeno linguistico in esame, in questo caso la *coreference*

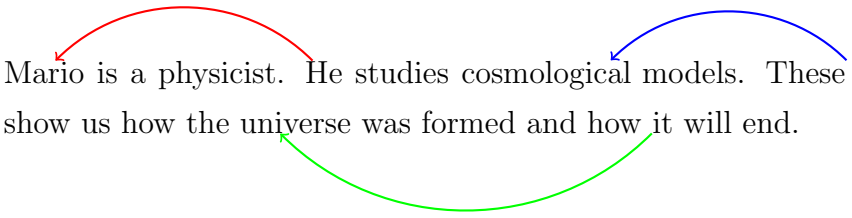
1.4 Coreference e Anafora Resolution

In seguito ad una breve descrizione dei fenomeni di anafora e *coreference*, si procede ad una discussione di cosa si intenda con *anafora/coreference resolution*. Oltre a tale aspetto, che è l'oggetto di studi di questa tesi, si procederà a fornire una panoramica storica degli approcci utilizzati per risolvere tale problema. Tale discussione ci fornisce importanti informazioni su quali siano stati il progresso e l'evoluzione degli strumenti utilizzati. Verrà dedicato spazio anche alla storia della disponibilità e creazione di corpora, fondamentali in particolar modo per gli approcci statistici o basati su *machine learning* alla *coreference resolution*. In questo capitolo utilizzeremo il termine *coreference*, in quanto esso è più generale rispetto all'anafora, distinguendo solo quando necessario di quale dei due fenomeni si tratti.

1.4.1 Definizione di coreference resolution

La *coreference resolution* si occupa, dato un testo, di individuare tutte le coreferenze in esso presenti. Esistono tuttavia almeno due possibilità per formalizzare tale processo, a seconda di come si interpreta tale fenomeno.

Esempio 13:



Nell'esempio precedente si evidenzia, tramite gli archi, il concetto di catena di *coreference*. Ciascuna *referring expression* viene infatti collegata al suo antecedente. Ricostruire una *coreference chain* significa dunque, per ciascuna espressione, trovare il proprio antecedente nelle frasi precedenti.

Formalmente è necessario ottenere un insieme della forma:

$$antecedente(m) = \begin{cases} \text{a} & \text{se a è l antecedente piu vicino ad m} \\ \epsilon & \text{se m non ha alcun antecedente} \end{cases} \tag{1.1}$$

Dove m è una espressione, mentre ϵ denota l'assenza di antecedenti. Effettuare la *coreference resolution* risulta dunque equivalente ad applicare la funzione antecedente a tutte le espressioni del testo, ottenendo dunque un insieme di archi che collegano ciascuna *referring expression* al suo antecedente. Un altro approccio prevede invece di osservare il problema da un punto di vista insiemistico.

Esempio 14:
Mario is a physicist. He studies cosmological models. These models can show us how the universe was formed and how it will end.

Nell'esempio precedente vengono evidenziate le catene di *coreference*, che però vengono interpretate come elementi di un insieme. In questo caso è necessario individuare l'insieme:

$$C = \{B | B \subseteq A \wedge \forall m_i, m_j \in (B), coreferents(m_i, m_j)\} \tag{1.2}$$

Dove:

- A è l'insieme delle espressioni del testo;
- coreferents(m,n) è una funzione che determina se due espressioni siano coreferenti.

Individuare C risulta dunque equivalente a risolvere il problema della *coreference* in un testo. Sebbene i due approcci siano equivalenti nel loro risultato finale, è evidente come l'approccio insiemistico suggerisca l'utilizzo di un algoritmo di clustering basato sulla *coreference*.

Un ulteriore aspetto riguardante la *coreference* è quello della cosiddetta *mention extraction*. È infatti evidente come sia auspicabile una la riduzione

del numero di espressioni considerate. Vi sono pertanto algoritmi ed approcci che consentono di ridurre il numero di candidati prima di applicare l'algoritmo di *coreference resolution*, scartando le espressioni che non codificano alcuna entità. Il tempo di esecuzione viene in questo modo ridotto in modo sostanziale e anche la possibilità di errori viene drasticamente limitata.

1.5 Approcci euristici alla coreference resolution

Una discussione, seppur incompleta, della *coreference resolution*, non può prescindere da una descrizione degli approcci che sono stati utilizzati nel tempo per affrontare il problema. Si è già evidenziato come la disponibilità di corpora sia un aspetto fondamentale rispetto alla possibilità di sviluppo di tecnologie per la *coreference resolution*. Si possono infatti individuare due approcci differenti alla risoluzione delle *coreference*: un approccio euristico ed un approccio statistico. Il primo approccio non necessita di corpora annotati, si basa invece sugli studi linguistici che sono stati brevemente discussi e su tecniche euristiche che sfruttino tali informazioni. L'approccio statistico richiede invece una grande mole di dati per essere efficace, in quanto utilizza i dati per approssimare tramite apprendimento automatico la risoluzione del problema. Seppure vi siano delle distinzioni, i due approcci si sono susseguiti temporalmente, in quanto la disponibilità di corpora era pressoché nulla quando i primi risolutori sono stati proposti. In anni più recenti si sono diffusi strumenti basati su apprendimento, che sfruttano appieno la disponibilità di corpora annotati.

Rispetto agli approcci euristici, forniremo una breve panoramica, descrivendo in dettaglio alcuni algoritmi che sono rappresentanti di altrettanti approcci differenti al problema.

1.5.1 L'algoritmo naive di Hobbs

Si è discussa in precedenza l'importanza di vincoli e preferenze di tipo morfo-sintattico rispetto al problema della *coreference*. Il precursore di algoritmi che utilizzano questo tipo di approccio basato su sintassi è l'algoritmo di Hobbs(Algoritmo 1) per la risoluzione dei pronomi, proposto negli anni '70 [28]. Tale algoritmo è di significativa importanza storica e viene spesso utilizzato tuttora come *baseline*.

Esso procede attraversando il parse-tree tramite un algoritmo *breadth-first*, da sinistra a destra, andando poi all'indietro nel documento una frase alla volta, alla ricerca di un antecedente del pronome in accordo ad esso per quel che riguarda il genere e il numero. Successivamente, l'algoritmo si assicura che non sia possibile ottenere una *Noun Phrase*[2] (NP) con lo stesso *binding domain* del pronome: nell'esempio "*John Likes him*" non è possibile che "*John*" sia l'antecedeente di "*him*". Il terzo passaggio impone che vi sia almeno un nodo di tipo NP o S fra il candidato e la radice. Grazie all'utilizzo di un appoccio *breadth-first*, l'algoritmo ha una preferenza intrinseca per i nodi più vicini alla radice dell'albero e per i nodi più a sinistra.

L'autore fornì all'epoca valutazioni del suo algoritmo che venne testato su testo con annotazioni sintattiche perfette, ottenendo ottimi risultati, con accuratezza pari al 88,3%. Test successivi hanno ridimensionato queste prime valutazioni, mantenendo però un valore di accuratezza molto alto, pari al 82% [36].

Algoritmo 1 Hobbs' Algorithm

1: **function** HOBBS(p)
2: $s_n \leftarrow$ the first NP dominating the pronoun p
3: $c_n \leftarrow$ the first NP or S up from s_n
4: $p \leftarrow$ the path from s_n to c_n

[2]Una Noun Phrase è una frase che svolge il ruolo sintattico di un nome, composta dal nome stesso e dai suoi eventuali modificatori

5: $retval \leftarrow$ the first NP found in the left of path p in a left-to-right bfs fashon that has an NP or S between iself and c_n or ϵ if none was found
6: **if** $retval \neq \epsilon$ **then**
7: return $retval$
8: **end if**
9: **if** $c_n$is the highest node in the sentence **then**
10: Look for candidates in the tree of previous phrases, breadth-first, left-to right.
11: **end if**
12: $l_n \leftarrow$ the first NP or S node up from l_n
13: $p \leftarrow$ the path between c_n and l_n
14: **if** l_n is an NP and the path p did non pass through the node that X immediately dominates **then**
15: return l_n
16: **end if**
17: $retval \leftarrow$ The first NP found traversing all branches below l_n to the left of p, ϵ if none was found
18: **if** $retval \neq \epsilon$ **then**
19: return $retval$
20: **end if**
21: **if** l_n is an S node **then**
22: $retval \leftarrow$ The first NP found traversing all branches below l_n to the right of p, not going below any NP or S node, ϵ if none was found
23: **end if**
24: **goto** 4
25: **end function**

1.5.2 Approcci basati sul senso comune

L'algoritmo di Hobbs fu precursore di una serie di algoritmi *knowledge-poor*, che utilizzassero quindi solo gli aspetti morfosintattici del problema per ri-

solvere le anafore o le *coreference*. Vi è una classe di algoritmi che utilizzano invece la conoscenza di senso comune per risolvere i problemi. Lo stesso Hobbs fu autore di quella che è nota come la sua ipotesi [27], che ritiene impossibile la risoluzione dei problemi di anafora senza l'utilizzo di informazioni sul senso comune.

Fra gli algoritmi basati su senso comune vi è *Deep Semantic Processing*, sviluppato da Charniak [12]. Tale algoritmo suppone di operare sull'output di un interprete semantico. Tramite inferenza deduttiva esso sarebbe dunque in grado di risolvere la *coreference* come effetto collaterale. L'algoritmo, a causa dei requisiti per ora irrealizzabili e di alcune problematiche legate al suo funzionamento, non fu mai realizzato completamente, ma risulta essere il primo tentativo sistematico di risoluzione dell'anafora tramite inferenza.

La più sistematica soluzione basata sull'utilizzo dell'inferenza fu proposta da Hobbs in [27]. Al fine di comprendere tale proposta è utile utilizzare un esempio simile a quello utilizzato per illustrare preferenza basata su senso comune nell'anafora.

> **Esempio 15:**
> The city council refused the women a permit because they feared violence;
> The city council refused the women a permit because they advocated violence;

Per Hobbs e più recentemente nella trattazione di Kehler et al [32] il connettivo *because* ha una precisa connotazione semantica, che viene chiamata *Explanation*. Se chiamiamo S1 la prima parte della frase ed S2 la seconda parte (variabile), possiamo esprimere la *explanation* come:
Explanation: viene dedotto P da S1 e Q da S2, dove $Q \rightarrow P$
Perché la giustificazione di S1 in termini di S2 sia corretta, è necessario che vi siano un assioma che afferma:

$$fear(X,V) \wedge advocate(Y,V) \wedge enable\_to\_cause(Z,Y,V) \rightarrow deny(X,Y,Z) \tag{1.3}$$

Tale assioma codifica il fatto che se X (il consiglio comunale) teme V (la

violenza), Y (i manifestanti) sostengono la violenza e Z (il permesso) permette a Y di causare V, è plausibile dedurre che X possa negare Z a Y. Un sistema in grado di ottenere i predicati logici necessari, di stabilire una relazione di *explanation* ed effettuare il *binding* delle variabili alle entità necessarie sarebbe in grado di risolvere l'anafora dell'esempio. Un tale sistema tuttavia richiederebbe la codifica di tutti gli assiomi come quello qui evidenziato, in grado di fornire regole per risolvere le *explanation*. Questo approccio tuttavia non risolverebbe l'esempio 1.3, in quanto la spiegazione del perché i docenti possano aver convocato gli studenti non è facilmente codificabile tramite assiomi.

1.5.3 Salience

Un'altra possibilità nell'ambito di soluzioni *rule-based* riguarda l'utilizzo di *salience*. Sebbene si sia visto come non sia sufficiente selezionare il candidato più prossimo come antecedente di una certa *referring expression*, l'utilizzo della *salience* è un approccio interessante, come del resto dimostrato anche dai buoni risultati dell'algoritmo di Hobbs, che seleziona gli antecedenti privilegiando i più prossimi al pronome. Fra le proposte più interessanti vi è SPAR di Carter [11], a sua volta basato sull'algoritmo di Sinder [55].

L'autore definisce come "*shallow process hypothesis*" la volontà di limitare al minimo l'utilizzo di strumenti *common sense based*, che risultano essere costosi e inaffidabili. Il sistema procede estraendo interpretazioni formali da una frase, nella quale le anafore sono lasciate irrisolte. Viene poi invocato il sistema di Sinder con alcune modifiche per risolvere ciascuna anafora. Tale sistema utilizza una rete semantica delle entità ed una struttura dati per tenere traccia di quali entità siano più in focus all'interno del documento esaminato. Tali informazioni vengono poi utilizzate da Sinder per ottenere strutture dati riguardanti varie tipologie di focus del discorso. Vengono poi fornite regole per l'interpretazione dei vari tipi di focus per tipologie diverse di espressioni anaforiche.

Una volta invocato il metodo di Sinder, SPAR calcola dei punteggi associati a ciascuna interpretazione semantica, a seconda di quante anafore siano state risolte. Vengono poi applicati alcuni vincoli e mantenuti i risultati dei paragrafi con punteggi più alti. Viene infine utilizzato il sistema di inferenza causale di Wilks [60, 62] basato su senso comune per modificare i punteggi così ottenuti. Se vi sono ancora dei punteggi pari, vengono utilizzate altre euristiche per effettuare un *tie-break*. Il sistema SPAR è stato testato sia con brevi frasi ottenute da Carter stesso, risolvendo tutti i pronomi anaforici, sia su testi scritti da altri, ottenendo una precisione del 93%.

Nell'ambito di algoritmi basati su *salience*, vi sono poi una serie di proposte basate sulla teoria del *centering*. L'algoritmo di Brennan, Friedman e Pollard [9], fra gli altri, propone di generare una serie di possibili interpretazioni sul focus del testo tramite la teoria del *centering*. Esse vengono elaborate tramite un algoritmo di tipo *generate-filter-rank*, che dapprima genera tutte le possibili coppie per una certa menzione, poi le filtra tramite vincoli di vario tipo, infine effettua un *ranking* utilizzando informazioni sul *center* durante il passaggio fra una menzione e quelle precedenti. Seppure tale approccio sia di sicuro interesse e basato su forti evidenze empiriche, i risultati di test indipendenti ed estensivi [56] forniscono risultati inferiori in modo statisticamente significativo a quelli ottenuti tramite l'algoritmo di Hobbs.

Un ulteriore approccio basato su *salience* sfrutta una definizione di *salience* detta *activation-based*. Invece di fornire un insieme di "foci", vengono assegnati valori a tutte le entità del discorso. Tali valori dipendono ad esempio dal ruolo grammaticale dell'entità, dalla sua distanza dalla menzione corrente. Nella proposta di Lapin e Lass [36] a ciascuna entità viene associato un valore che misura il suo grado di *salience* ed esso viene modificato dopo ciascuna menzione nel testo. L'algoritmo opera poi tramite un sistema *generate-filter-rank* e sfrutta le informazioni discrete sulla *salience* per ipotizzare gli antecedenti del pronome anaforico corrente. I test sperimentali effettuati su un corpus di 360 esempi di pronome anaforico tratti da manuali

specialistici su computer, fornisce un'accuratezza dell'86%, rispetto all'82% dell'algoritmo di Hobbs applicato agli stessi dati.

1.5.4 Heuristic Based

Attorno al 1995 vi fu un cambiamento importante nel panorama della risoluzione dell'anafora. Tale periodo viene definito da Poesio et al. [48] come *robustiphication phase*. Se nel periodo precedente gli algoritmi venivano valutati su corpora di piccole dimensioni ed assumevano quasi sempre di avere accesso a informazioni corrette per quanto riguarda il parsing e le informazioni di senso comune, successivamente le soluzione proposte vengono progettate per tener conto di informazioni spurie e per essere applicabili a corpora più estesi.

Gli algoritmi emergenti in questo periodo si basavano su metodi euristici per operare in un contesto *knowledge-poor*. Fra le varie proposte gli algoritmi cogNIAC [7] e MARS [45] per la risoluzione dei pronomi risultano essere fra i più influenti. Alla base di coGNIAC e MARS è la supposizione che l'ipotesi di Hobbs non sia valida o per lo meno eccessivamente pessimista: si afferma infatti che non siano necessarie informazioni sul senso comune per risolvere almeno una sottoclasse delle anafore pronominali. L'algoritmo coGNIAC fu il primo a sfruttare un approccio definito "*precision first*", sfruttando sei regole, ordinate secondo la loro affidabilità su un set di 200 pronomi di training, e scartando le decisioni che non forniscono una confidenza sufficiente. Le valutazioni di tale algoritmo ottengono risultati simili a quelli dell'algoritmo naif di Hobbs. L'approccio utilizzato da MARS è basato su *generate-filter-rank*. Vengono dapprima identificati i pronomi, si usano poi regole sintattiche per ottenere i possibili antecedenti di ciascun pronome, si ordinano infine i candidati secondo 14 indicatori. Valutando tale algoritmo su un corpus di pronomi derivanti da manuali tecnici si ottenne un accuratezza del 61,6%.

1.5.5 Definite descriptions: l'algoritmo di Vieira e Poesio

Gli algoritmi descritti in precedenza sono per la maggior parte dedicati alla risoluzione di anafore pronominali, mentre le anafore nominali generiche, che costituiscono un problema più complesso, non vengono considerate.

L'algoritmo di Viera e Poesio [49], che si concentra sulla risoluzione delle descrizioni definite, risulta essere pertanto una proposta interessante. Le descrizioni definite sono espressioni del tipo:

> **Esempio 16:**
> For the first time John saw Venice appear in the distance. The city was barely visible in the fog, but it's charm was still noticeable despite the climate.

È evidente come l'espressione "*the city*" sia molto più espressiva di una forma pronominale. Oltretutto, è interessante evidenziare come il riconoscimento di pronomi non anaforici, come "*it*" nella sua forma impersonale in inglese sia di più facile descrizione rispetto a forme anaforiche di descrizioni definite. Tali forme infatti possono essere espressioni "*discourse new*", ovvero che si riferiscono ad un'entità per la prima volta nel testo in esame.

L'algoritmo opera attraverso tre funzioni distinte: vengono dapprima individuate le espressioni non anaforiche (*discourse-new*), si cercano antecedenti "*same-head*", che cioè condividono con la *referring expression* la *head* [3], si cercano dunque *bridging description*. In generale, la discriminazione fra espressioni anaforiche ed espressioni *discourse-new* è un problema di complessa risoluzione, la cui discussione completa non è parte di questa tesi. Gli autori propongono tuttavia un approccio che, tramite un dizionario di dimensioni contenute e un'euristica, è in grado di riconoscere alcune classi di espressioni non anaforiche. Vengono ad esempio considerati non anaforici i predicati funzionali con un complemento, come ad esempio:

[3] In linguistica si intende per *head* la parola che determina la categoria sintattica della frase (eg la *head* di "bella giornata" è "giornata")

Esempio 17:
John left his house despite the fact that it was snowing.

In questo contesto l'espressione "*the fact*" viene seguita da un complemento, che ne determina la non anaforicità. Altri termini non anaforici sono detti "l*arger situation uses*" (eg *the king*, *the moon*, etc) e determinare se essi siano anaforici richiede una conoscenza enciclopedica. L'approccio di Vieira e Poesio per la risoluzione di questi ultimi utilizza una breve lista di tali termini ed un'euristica per individuare *named entity* [4] (eg *the Waterloo battle*). Vengono infine considerati post-modificatori restrittivi ("*the car that he bought*") ed apposizioni ("*My sister, Alice*").

Una volta identificati i termini *discourse-new*, si cerca, per ciascuno dei termini che sono stati ipotizzati come anaforici, se vi siano degli antecedenti *same-head*. La distanza dall'espressione viene considerata, scartando candidati troppo distanti, considerando in questo caso solo menzioni successive della stessa entità ed espressioni che risultano identiche alla *referring expression*. Si applicano poi euristiche di compatibilità, con lo scopo di evitare situazioni in cui due espressioni che hanno descrizioni incompatibili siano considerate anaforiche ("*the new house*", "*the old house*"). Se alla fine del processo vi sono ancora più candidati possibili, viene scelto quello più recente.

Un ultimo caso riguarda le *bridging description*, ovvero i casi in cui vi è anafora nei casi di definite *description*, ma non vi sono termini in comune fra le due espressioni:

Esempio 18:
In "1984" George Orwell described the dangers of totalitarianism in a modern society and the disconnect from reality it caused in the conscience of the individual. It is widely believed that the author was inspired by Lev Trotsky for the character of Emmanuel Goldstein, the revolutionary leader in the book.

Per risolvere questo tipo di anafora, l'algoritmo utilizza sia informazioni su relazioni concettuali fornite da WordNet [44], sia un riconoscitore di *named*

[4] Si intende per named entity un nome proprio che si riferisce ad un'entità

entity, in grado cioè di riconoscere entità che vengono espresse tramite un nome proprio e di categorizzarle come persone, luoghi o organizzazioni.

Utilizzando le componenti descritte in precedenza, l'algoritmo analizza una frase alla volta, identificando tramite un euristica le *definite description*. Gli altri NP, con l'eccezione dei pronomi, vengono considerati come *discourse-new* e per ciascuno di essi viene costruita una *file-card*. Si discrimina dunque fra entità *discourse-new* e *discourse-old* tramite un albero di decisione, inserendo le espressioni *discourse-old* nel *file-card* dell'antecedente. Vengono poi utilizzate ulteriori euristiche per risolvere le *bridging descriptions*.

I *decision tree* ed i parametri utilizzati dalle varie euristiche sono stati tarati tramite analisi su dati reali in modo da massimizzare la loro efficacia su un corpus di circa 1000 *definite description* annotate per il *training*, 400 per il *testing*. L'algoritmo nella sua iterazione migliore ha fornito un'accuratezza ed un *recall* del 77%.

1.5.6 Sistemi euristici nei task MUC 6 e MUC 7

Le proposte finora analizzate, seppure efficaci nel loro dominio specifico, si limitavano a fornire una soluzione ad una piccola parte del problema della *coreference*, ad esempio a risolvendo solo i pronomi anaforici o le *definite description*. Nel 1996 venne però introdotto, nell'ambito della sesta edizione della *Message Understanding Conference* (MUC). un task di *coreference resolution* generica. Tale task richiedeva ai partecipanti di costruire un sistema che fosse in grado di risolvere il problema della *coreference* nella sua interezza, un compito molto più complesso rispetto a quelli di dominio specifico affrontati fino ad allora. La creazione di tale task fu strumentale allo sviluppo di sistemi che integrassero gli studi dei decenni precedenti, nonché a fornire corpora standard per la valutazione delle performance di sistemi di *coreference resolution*.

Fra i sistemi che meglio si comportarono nei task vi sono FASTUS [4] e LaSIE [23], che risultarono essere i migliori partecipanti rispettivamente in MUC 6 e MUC 7.

FASTUS fu sviluppato per la conferenza MUC 6 e risulta essere un sistema *rule-based* che sfrutta un una grammatica a stati finiti. Esso procede effettuando un'analisi per *chunk* sul testo, che produce risultati molto accurati per porzioni di testo contenenti nomi complessi, ma che non produce una struttura gerarchica come quelle richieste dalle soluzioni viste in precedenza, che presupponevano la disponibilità di un'analisi sintattica completa. L'input dell'algoritmo sono le menzioni che forniscono già l'estensione dell'espressione da considerare, nonché informazioni linguistiche di vario tipo (numero, genere, classe semantica di appartenenza,etc). Vengono dunque applicate regole differenti a seconda del tipo di anafora che si vuole risolvere:

- **Pronomi**: viene forzato un limite di tre frasi per la ricerca dell'antecedente, nonché vincoli sintattici e semantici. I candidati vengono poi ordinati secondo un algoritmo di *salience ordering* sfruttando le informazioni ottenute dalla grammatica a stati.

- **Definite Noun Phrases**: la finestra di ricerca è pari a 10 frasi, mentre non viene limitata per i nomi propri. Viene poi utilizzata una *sort consistency*, che si assicura che l'ordine delle espressioni sia corretto, consentendo ad esempio una catena fra "l'Italia" e "il paese" ma impedendo la stessa catena con l'ordine delle espressioni invertito. Vengono inoltre eliminate coppie anafora-antecedente per coppie che hanno modificatori incompatibili, come nel caso di "*the british man*" e "*the french man*". Per i nomi propri viene inoltre utilizzata un'euristica che riconosca la classe semantica cui appartengono.

Nel task di *coreference* di MUC 6 FASTUS risultò come il miglior partecipante, con *recall* del 59% e precisione del 72%.

Il sistema LaSIE ed il suo successore LaSIE-II [31] sono due sistemi non unicamente sviluppati per la *coreference resolution*, includono infatti altre componenti di un sistema di elaborazione del linguaggio naturale. Rispetto alla *coreference*, essi tentano di costruire un modello del discorso completo, nel quale si collega ciascuna menzione all'entità che essa esprime, espandendo

poi tali informazioni per costruire un'ontologia completa a partire dalle entità del testo.

1.5.7 Approcci euristici moderni

Nonostante il successo di proposte per la risoluzione della *coreference* basate su apprendimento, sono ancora oggetto di studi approfonditi sia approcci puramente euristici, come lo Stanford Deterministic Coreference Resolution System [38], sia sistemi ibridi, che sfruttano sia componenti euristiche che apprendimento. Sebbene sia opinione dell'autore che lo sviluppo di tecnologie basate su learning costituisca la direttrice preferenziale per lo sviluppo di strumenti per risoluzione della *coreference* e in generale per affrontare molti problemi legati all'elaborazione del linguaggio naturale, è indispensabile durante lo studio del problema un'analisi approfondita delle soluzioni proposte, anche se non più competitive nell'epoca contemporanea. Tramite tale studio, infatti, è possibile individuare le problematicità che sono emerse e le soluzioni adottate nel tempo, in modo da poter incorporare gli aspetti tuttora rilevanti di tali studi. Sono oltretutto indispensabili studi approfonditi su sistemi ibridi, che sfruttino sia i vantaggi di un approccio rigoroso, che tenti di incorporare i risultati degli studi linguistici approfonditi sul fenomeno, sia gli innumerevoli vantaggi di un sistema che sia in grado di apprendere le caratteristiche del fenomeno analizzato, consentendo una generalizzazione spesso di complessa traduzione attraverso un algoritmo imperativo.

Una breve discussione è necessaria per quanto riguarda la determinazione dell'anaforicità in un documento. Tutti gli algoritmi basati sui paradigmi seguenti, infatti, considerano di operare solo su menzioni anaforiche, mentre la risoluzione del problema richiede che si individuino le menzioni anaforiche o coreferenti. Per tale ragione le valutazioni degli algoritmi basati sui paradigmi seguenti hanno spesso utilizzato *mention extractor* per valutarne l'efficacia anche con menzioni imperfette.

1.6 Modelli per il machine learning nella coreference resolution

Successivamente ad una prima fase durante la quale non vi era disponibilità di corpora annotati per la risoluzione del problema della *coreference*, si sono susseguiti una serie di task, che hanno consentito sia di stabilire quali fossero le metriche più accurate per misurare l'efficacia dei sistemi proposti, sia di costruire corpora annotati in modo da consentire uno sviluppo di sistemi basati su machine learning.

In questa sezione discuteremo brevemente dei vari approcci basati su *machine learning*, in particolar modo per quel che riguarda i paradigmi di maggior successo. Non sarà tuttavia possibile discutere di approcci basati su reti neurali *deep*, che pure sono state oggetto di studio approfondito durante lo sviluppo del risolutore proposto in questa tesi. Tali algoritmi verranno invece descritti successivamente, una volta che si saranno discusse le principali componenti delle reti neurali, in modo da consentire una migliore comprensione di tali risolutori.

1.6.1 Il modello mention-pair

Grazie al consolidamento di corpora annotati, è stato possibile lo sviluppo di sistemi basati su *learning*. Fondamentale da questo punto di vista è l'approccio scelto, ovvero il tipo di output che si intende ottenere a partire dai *solver* basati su *machine learning*. Si è discusso in precedenza (sezione 1.4.1) come dal punto di vista astratto, la risoluzione di una *coreference* possa essere espressa in due modi distinti: si possono esprimere le *coreference chain*, come una lista di collegamenti fra espressioni coreferenti successive, oppure immaginarle come un insieme di menzioni coreferenti. Tali interpretazioni, seppure non suggeriscano una strategia di risoluzione, forniscono indicazioni su come si possano costruire algoritmi basati su *machine learning*.

Da questo punto di vista al modello basato su archi fra *referring expression* ed antecedenti corrisponde il *mention-pair model* [3]. Tale modello

trasforma il problema della *coreference* nel trovare l'antecedente m_j della referring expression m che soddisfi:

$$\arg\max_{m_j} P(C(m_j, m)) \tag{1.4}$$

Dove si intende individuare la menzione m_j che massimizza la probabilità, stimata dal sistema, che la menzione corrente m sia coreferente con essa. Nelle implementazioni basate su *mention-pair model*, le due menzioni vengono rappresentate da strutture dati contenenti informazioni di varia natura, quali distanza fra esse, parte del discorso, etc. Tramite tali informazioni è possibile discriminare fra coppie coreferenti e non coreferenti e costruire un sistema che sia in grado di apprendere tale differenza e generalizzarla. Una volta che si sono costruite le valutazioni della coreferenza fra coppie di menzioni, è necessario procedere con un passaggio di *clustering* per costruire le entità su tutto il documento. Da questo punto di vista, è possibile utilizzare varie strategie. Rispetto al *clustering*, una strategia *closest-first* opera a ritroso selezionando sempre il candidato più vicino, mentre *best-first* seleziona la menzione con la maggior probabilità di essere antecedente dell'espressione corrente. Un'ultima possibilità, infine, riguarda *aggrssive merge*, che effettua il *clustering* di tutte le menzioni che vengono valutate come coreferenti rispetto all'espressione corrente.

Se il modello è di certo interesse, è pur vero che vi sono una serie di problemi nella sua formalizzazione. È infatti evidente come la *coreference* sia un problema che non riguarda solo due menzioni all'interno del testo, ma risulta essere molto più ampio, se non addirittura avere un'orizzonte globale rispetto al testo. Riducendo il problema ad un semplice assegnamento di verità per ciascuna coppia di elementi, potremmo ottenere risultati inaspettati:

Esempio 19:
Clinton$_{\text{A}}$ was a president of the US$_{\text{A}}$ between 1993 and 2001. He$_{\text{A}}$ then had to resign [...] [Hillary] Clinton$_{\text{B}}$ was a presidential candidate$_{\text{B}}$ in 2009 and 2016. She$_{\text{B}}$ won the democratic primary.[...] Donald Trump$_{\text{C}}$ won the

> election and became the 45th US president$_{C}$.

È evidente nell'esempio precedente come una scelta binaria fra due menzioni possa condurre ad assegnamenti di coreferenza incompatibili, in quanto è il contesto stesso delle frasi a suggerire a quale entità si riferiscano le espressioni riguardanti il presidente degli Stati Uniti. Se però la frase precedente è di complessa interpretazione, il modello *mention-pair* può condurre ad errori anche in situazioni più semplici, sempre però fortemente dipendenti dal contesto. È possibile, ad esempio, considerare coreferenti due nomi propri e lo stesso pronome, in quanto non vengono tenuti in conto i precedenti assegnamenti di coreferenza. La mancanza di contesto non è totalmente risolvibile modificando l'algoritmo di *clustering* né modificando l'insieme di *feature* considerate, in quanto il modello non prevede l'utilizzo di informazioni che riguardano l'intero cluster finora costruito.

1.6.2 Entity-mention model

A causa delle problematiche descritte in precedenza, una delle proposte fu quella di un modello detto *entity-mention model* [65]. Tale modello sfrutta, in modo analogo a quanto visto per *mention-pair model*, tre insiemi di *feature*. Tali *feature* riguardano la menzione corrente, il cluster precedente e la relazione fra i due. Una volta che l'algoritmo ha effettuato una decisione sulla coreferenza della menzione rispetto ai cluster precedenti, è possibile, come nel caso di *mention-pair*, utilizzare algoritmi di *clustering* differenti per risolvere la *coreference*. Rispetto a *mention-pair*, tuttavia, *entity-mention* viene applicato online, ovvero dopo ciascuna applicazione del classificatore ai dati, vengono modificati i cluster, che a loro volta sono parte dell'input durante l'analisi di menzioni successive. Nonostante le buone premesse, i risultati di *entity-mention* furono deludenti [40], tuttavia mostrarono come tale modella riusciva a mantenere una coerenza di cluster, ovvero a non costruire entità con assegnamenti di coreferenza *pairwise* non incompatibili ma globalmente errati, utilizzando solo una frazione delle *feature* richieste da *mention-pair*.

Malgrado i risultati scoraggianti ottenuti da *entity-mention* model, è evidente come quest'ultimo modello abbia caratteristiche interessanti, in quanto le informazioni globali riguardanti i cluster sono intuitivamente indispensabili per ottenere buoni risultati nella *coreference resolution*. Proprio per questo è a partire da tale modello che si sono sviluppate ulteriori proposte, in grado di migliorare le prestazioni del sistema. Una di queste proposte [19] opera, a differenza di *entity-mention* model, determinando se un insieme di menzioni sia coreferente o meno. Viene dunque applicato un algoritmo di *clustering* di tipo *greedy*, nel quale inizialmente ciascuna menzione è parte di un cluster contenente solo sé stessa, successivamente vengono uniti i cluster che vengono valutati come più probabilmente coreferenti. L'algoritmo procede finché non vi sono più cluster considerati coreferenti dall'algoritmo. Si noti come tale approccio corrisponda in maniera quasi esatta alla formalizzazione insiemistica discussa in 1.4.1 e trasformi la risoluzione della *coreference* in un problema di *clustering* vero e proprio.

1.6.3 Mention-Ranking e Cluster-Ranking

Un'altra proposta, detta *mention-ranking model* [16, 17], opera fornendo un *ranking* delle menzioni rispetto ad una menzione corrente, a seconda della probabilità stimata che essi siano coreferenti. Al fine di fornire questo *ranking*, l'algoritmo opera su due candidati possibili, selezionando quello che viene valutato come probabilmente più coreferente. In questo modo è possibile risolvere le *coreference* tramite vari algoritmi di *clustering*. È possibile utilizzare un torneo ad eliminazione, nel quale le menzioni "perdenti" vengono scartate e si seleziona come antecedente il candidato che vince il torneo. Un approccio alternativo, detto *tournament ranking*, confronta tutte le coppie di candidati della menzione corrente, selezionando il candidato che vince più frequentemente. Viene selezionato sempre il candidato più vicino alla menzione corrente per risolvere i pareggi.

Una ulteriore proposta riguarda il cosiddetto *cluster-ranking* model [52]. Come nel caso di *mention-ranking*, si opera su coppie di candidati, fornendo

una preferenza dell'algoritmo fra essi. I candidati, tuttavia, non vengono visti come menzioni, bensì considerati come cluster, quindi, in modo analogo a quanto visto con entity-model, essi vengono codificati attraverso *feature cluster-wise*.

Si sono esaminati brevemente i principali approcci per la *coreference resolution* mediante *machine learning*, evidenziando punti di forza e debolezze che le contraddistinguono. Si vedrà in seguito come tali proposte siano state valutate durante l'implementazione del nostro solver, valutandone l'efficacia rispetto all'insieme di *feature* selezionato ed infine giungendo ad una implementazione ibrida.

1.6.4 Integer Linear Programming

Uno degli approcci più efficaci per la risoluzione delle *coreference* utilizza *Integer Linear Programming* (ILP). Per ILP si intende un approccio basato su vincoli e funzione obiettivo, espressi in modo lineare, in quanto per questa sottoclasse di problemi espressi tramite vincoli è possibile utilizzare algoritmi di risoluzione molto efficienti, che minimizzino la funzione obiettivo e rispettino tutti i vincoli imposti. Rispetto alla *coreference* si utilizzano solitamente delle variabili aleatorie $X_{i,j}$, che esprimono la coreferenza fra coppie di menzioni tramite valori binari. Attraverso tale formulazione, è possibile fornire una probabilità del documento corrente D:

$$P(D) = \prod_{j=2}^{n} \prod_{i=1}^{j-1} P(X_{ij} = x_{ij}) \tag{1.5}$$

Tramite tale formulazione, poiché le variabili X_{ij} possono assumere solo valori 0 ed 1, è possibile esprimere il numero di predizioni corrette nel documento come:

$$C(D) = \sum_{j=2}^{n} \sum_{i=n}^{j-1} P(X_{ij} = x_{ij}) \tag{1.6}$$

Scegliendo una delle due formulazioni di funzione di costo, è possibile implementare modelli che sfruttino i vincoli di un sistema ILP ed un algoritmo di *learning* per risolvere il problema della *coreference* in modo efficiente.

Un esempio di tale approccio è la proposta di Bengston e Roth [8]. Tale algoritmo utilizza un approccio *pair-wise*, fornendo assegnamenti di verità per ciascuna variabile X_{ij} che rappresenta la coreferenzialità di due menzioni i,j. Una volta ottenuti tali assegnamenti di verità si utilizza un algoritmo di *clustering* "*best-link*". Rispetto alle coppie di menzioni, viene utilizzato l'algoritmo dell'*averaged percepron* [22] per fornire una funzione di score di due menzioni m_i, m_j:

$$c(m_i, m_j) = \sigma(\langle m_i, m_j \rangle) - \theta \tag{1.7}$$

Dove la funzione σ è definita come:

$$\sigma(\langle m_i, m_j \rangle) = \boldsymbol{w} \cdot \Phi(\langle m_i, m_j \rangle) \tag{1.8}$$

La funzione c viene interpretata nel modo seguente: viene creato un link fra m_i ed m_j sse $\sigma(\langle m_i, m_j \rangle) > \theta$. Tale vincolo è parte della funzione costo, che incoraggia il risolutore a creare un link solo se viene soddisfatta tale condizione. La funzione obiettivo è dunque la seguente:

$$max \sum_{j=2}^{n} \sum_{i=1}^{j-1} (\sigma(\langle m_i, m_j \rangle) - \theta) x_{i,j} \tag{1.9}$$

che opera dunque massimizzando il numero di link di coreferenza creati solo se la funzione c ha valore superiore a θ, poiché in caso contrario $c(m_i, m_j) < 0$ influisce negativamente sulla funzione costo. L'algoritmo è dunque spinto ad assegnare il valore 1 solo ad istanze positive. Mentre i vincoli espressi sono i seguenti:

$$\sum_{i=1}^{j-1} x_{i,j} \leq 1 \quad \forall j \tag{1.10a}$$

$$0 \leq x_{i,j} \leq 1 \quad \forall i,j | i < j \tag{1.10b}$$

$$x_{i,j} \in \mathbb{Z} \quad \forall i,j | i < j \tag{1.10c}$$

I tre vincoli esprimono dunque rispettivamente:

- Ciascuna menzione ha al più un antecedente;
- Ciascuna variabile è compresa fra 0 ed 1;
- Ciascuna variabile è intera.

Tramite questo semplice insieme di vincoli e la funzione obiettivo, si è espresso un semplice modello che fornisce però un *clustering* implicito fra menzioni: operando su coppie di menzioni, è infatti possibile che $X_{x,y} = 1 \wedge X_{y,z} = 1 \wedge X_{x,z} = 0$. È evidente come tale situazione crei dei problemi ad un passaggio di *clustering* successivo all'applicazione dell'algoritmo ILP, in quanto esso deve tener contro di assegnamenti di verità inconsistenti.

Per risolvere questo tipo di problemi, un approccio possibile è quello di esprimere in modo esplicito il *clustering*, attraverso un vincolo che espliciti la transitività della relazione di coreferenza. Una formulazione tramite ILP possibile è la seguente:

$$max \sum_{j=2}^{n} \sum_{i=1}^{j-1} c(m_i, m_j) x_{ij} \tag{1.11a}$$

$$x_{ij} + x_{ik} + x_{jk} \leq x_{ik} + 1 \quad \forall i,j,k | i < j < k \tag{1.11b}$$

$$x_{ij} + x_{ik} + x_{jk} \leq x_{jk} + 1 \quad \forall i,j,k | i < j < k \tag{1.11c}$$

$$x_{ij} + x_{ik} + x_{jk} \leq x_{ij} + 1 \quad \forall i,j,k | i < j < k \tag{1.11d}$$

$$0 \leq x_{i,j} \leq 1 \quad \forall i,j | i < j \tag{1.11e}$$

$$x_{i,j} \in \mathbb{Z} \quad \forall i,j | i < j \tag{1.11f}$$

Che include, oltre ai vincoli già visti in 1.10, i tre vincoli necessari per la transitività Si noti come tale formulazione includa solo il numero minimo di

tali vincoli, a differenza della forma:

$$x_{ij} + x_{jk} \leq x_{ik} + 1 \quad \forall i, j, k \tag{1.12}$$

che, se appiattiamo la forma $\forall$ per una certa tripla di i, j, k, contiene forme ridondanti del tipo:

$$x_{12} + x_{23} \leq x_{13} + 1 \quad \forall i, j, k \tag{1.13a}$$

$$x_{21} + x_{23} \leq x_{13} + 1 \quad \forall i, j, k \tag{1.13b}$$

dove se assumiamo un ordinamento fra le menzioni che corrisponda ai propri indici, otteniamo una variabile X_{21} che non ha significato, in quanto il sistema deve tener conto dell'ordine delle menzioni, considerando solo variabili della forma $X_{ij}|i < j$.

Oltre alle semplici formalizzazioni già fornite, altre soluzioni in letteratura propongono di modificare l'approccio *best-link* creando variabili ausiliarie che tengano traccia di quali entità siano collegate tramite una catena di coreferenza. Questo tipo di variabili è utile per formalizzare tipologie di vincoli su cluster di menzioni, in modo da gestire, ad esempio, i casi di incompatibilità di *join* fra cluster multipli di menzioni.

Altre soluzioni utilizzano invece modelli multipli che si occupano della espressione di vincoli riguardanti aspetti diversi del problema. Si procede definendo tre modelli, che riguardano la determinazione di quali termini siano anaforici, la classificazione di *named entity* rispetto ad alcune categorie (es nel caso del dataset ACE: persona, organizzazione, luogo, entità geopolitica, edificio), nonché chiaramente la coreferenza fra menzioni. Un tale sistema risulta essere molto efficace rispetto ad approcci *best-link* semplici, in quanto è in grado di utilizzare molte più informazioni e fornire una soluzione completa di *coreference resolution*.

Rispetto alle proposte viste in precedenza, gli approcci basati su ILP risultano essere molto efficaci nella *coreference resolution*, in quanto permettono di utilizzare algoritmi di *learning* che operano semplicemente su coppie

di menzioni, mantenendo tuttavia la correttezza complessiva della soluzione e addirittura integrando la discriminazione dei termini anaforici [20, 21].

Si tenga presente che esistono approcci alla *coreference resolution* basati sull'utilizzo di reti neurali, la cui descrizione verrà tuttavia presentata nella sezione 2.2, dopo aver descritto i meccanismi alla base del *deep learning*, in modo da consentie al lettore di meglio comprendere gli strumenti che sono alla base di tali *solver*.

1.7 Risorse, Metriche ed utilizzi della Coreference Resolution

Nelle sezioni precedenti ci si è concentrati sulla formalizzazione della *coreference*, sugli studi linguistici riguardanti tale fenomeno e sulle soluzioni proposte per la *coreference resolution*. Vi sono tuttavia altri aspetti fondamentali che non possono essere omessi da una trattazione, per quanto sintetica, della *coreference resolution*. È infatti indispensabile una trattazione dei principali corpora annotati disponibili, senza i quali gli approcci basati su *machine learning* non sarebbero possibili. È inoltre necessario stabilire delle metriche adeguate alla valutazione di un sistema di *coreference resolution*, in quanto vi sono svariati aspetti di cui tener conto e non vi è una metrica che sia migliore da ogni punto di vista rispetto alle altre. Un ultimo aspetto riguarda le possibili funzioni che possono essere svolte da un sistema di *coreference resolution*: anche assumendo la possibilità di implementare un solver perfetto, che sia in grado di risolvere il problema nella sua interezza, è lecito chiedersi se l'attenzione a questo problema sia di interesse puramente linguistico o se un solver possa avere altri ruoli in una pipeline di un sistema di elaborazione del linguaggio naturale più complesso. Saranno quindi questi gli argomenti di cui ci occuperemo in maniera sintetica, al fine di completare la trattazione della *coreference resolution*.

1.7.1 Corpora per la Coreference Resolution

Se i primi sistemi per la *coreference resolution* non ebbero la possibilità di utilizzare corpora standard, valutando invece le proprie performance su *dataset* di dimensioni contenute e solitamente compilati manualmente dagli autori, sono state le campagne di valutazione a fornire corpora standardizzati in modo da comparare le performance di sistemi diversi. Tali corpora, oltre ad essere un prerequisito fondamentale per l'emergere di sistemi basati su *machine learning*, che richiedono *dataset* di dimensioni relativamente estese, sono stati strumentali alla ridefinizione del problema. Furono infatti le conferenze MUC-6 (1996) e MUC-7 (1997) a spostare il focus dei ricercatori da alcuni aspetti del fenomeno della *coreference*, quali i pronomi anaforici o le descrizioni definite, alla risoluzione di tale problema nella sua interezza, almeno per ciò che concerne le anafore nominali.

I corpora costruiti per i task MUC-6 e MUC-7 furono i primi a contenere annotazioni per il training e la valutazione di sistemi di *coreference resolution*. Il *dataset* MUC-6 è composto da 25 articoli del Wall Street Journal riguardanti dispute nell'ambito lavorativo e successioni alla guida di aziende, composti da un totale di 30000 parole. MUC-7 è un *dataset* di dimensioni simili, composto da documenti riguardanti incidenti aerei e lanci di razzi. Le *guideline* stabilite per l'annotazione di tali corpora fu molto influente e si basa sulla coreferenza fra NP. I casi di menzione nominale di entità del discorso furno gli unici considerati, mentre vennero escluse relazioni nelle quali l'anafora o l'antecedente sono introdotti esplicitamente come parte di un NP. MUC-6 e MUC-7 furono i primi corpora annotati per la *coreference*, ma non sono più utilizzati se non per comparazioni con sistemi valutati su tali *dataset*.

A seguito delle campagne MUC, venne creata la campagna di valutazione Automatic Content Extraction (ACE), che si concentrava sull'estrazione di informazioni da documenti scritti. Le campagne di valutazione ACE si tennero fra il 2000 ed il 2008, ma furono i corpora in lingua inglese creati per i task di *Entity Detection and Tracking* di ACE-2 ed ACE-2005 a sostituire

MUC come standard per la *coreference resolution*. In questi corpora, a differenza dei corpora MUC, l'annotazione delle entità è limitata a precise classi semantiche: inizialmente persone, organizzazioni, luoghi, entità geopolitiche, armi, veicoli. Le linee guida seguite da ACE furono estese rispetto a quelle di MUC a causa dell'inclusione di corpora in altre lingue (cinese,arabo) ma mantennero i principi generali usati da MUC.

Negli anni successivi vennero proposte delle linee guida nell'ambito dell'iniziativa DRI e del progetto MATE [47], che vennero adattate per la creazione di corpora in varie lingue. In particolare per l'inglese si ebbero OntoNotes, GNOME ed Arrau, mentre vennero compilati corpora in varie altre lingue: fra cui CORREA per l'olandese, ANCORA per lo spagnolo ed il catalano e LiveMemories per l'italiano. Tali linee guida sono maggiormente ispirate da studi linguistici, per cui vengono annotati tutti gli NP, senza tener conto della loro classe semantica, e le menzioni tendono a corrispondere agli NP individuati durante l'analisi sintattica.

Il corpus OntoNotes, che viene utilizzato dal nostro solver, è il *dataset* di dimensioni più ampie in lingua inglese, in quanto include 1,6 milioni di parole. Vi sono anche corpora in lingua cinese (1M di parole) ed un corpus meno esteso in lingua araba (300k parole). Il corpus in lingua inglese comprende documenti provenienti da agenzie di stampa, notiziari televisivi, trasmissioni televisive, pagine web, conversazioni telefoniche. Dal punto di vista della *coreference*, vengono annotati tutti i casi di *coreference* anaforica, senza restrizioni di classi semantiche e non limitandosi ai soli NP. Vengono individuati due tipi di *coreference*, identico (marcato con IDENT) ed appositivo (marcato con APPOS), in modo da consentire una trattazione distinta dei due fenomeni. Dal punto di vista delle linee guida, esse sono ispirate a MUC ed ACE, in modo consistente alle idee di IDEE e MATE. Parte di OntoNotes venne utilizzata nella campagna di valutazione SemEval 2010, che comprendeva un task di *coreference resolution*, che ebbe il grande merito di unificare in un formato standard corpora in varie lingue.

Si sono fornite in maniera estremamente sintetica le principali risorse

per la valutazione standard di sistemi per *coreference resolution*. Il nostro sistema, in particolare, sfrutta il corpus OntoNotes che è stato convertito nel formato utilizzato dal task di *coreference* di CoNLL 2012, utilizzato ad oggi come standard de facto per la valutazione di tutti i sistemi di questo tipo.

1.7.2 Metriche per la Coreference Resolution

Un'analisi superficiale della *coreference resolution* potrebbe indurre a ritenere tale problema assimilabile ad un qualsiasi problema di *clustering*, utilizzando dunque metriche derivanti da questo tipo di task. Tuttavia è evidente come vi sia un problema fondamentale derivante da questo assunto: a differenza degli algoritmi di *clustering* generici, che operano su oggetti forniti come input all'algoritmo e quindi corretti per definizione, la maggior parte dei sistemi di *coreference* opera su menzioni rilevate da un sistema automatico, che sono quindi soggette ad errore. A tal fine sono state proposte varie metriche, fra le quali non vi è di fatto una soluzione preferita in letteratura rispetto alle altre.

Al fine di esplicitare le metriche proposte, è necessario introdurre alcune notazioni standard, che viene tratta da "Anaphora Resolution" [48] e fornisce una formalizzazione di tali metriche con una notazione comune. In particolare, sono fondamentali i concetti di menzione e di entità, che possono entrambi essere annotazioni esatte (*key*) oppure estratti dal solver. L'insieme delle *key entities* è definito come $\mathscr{K} = \{K_i\}_{i=1}^{|\mathscr{K}|}$, dove K_i è un insieme di menzioni coreferenti fra loro che costituiscono l'i-esima entità. In modo analogo definiamo l'insieme delle *response entities*, ovvero le entità individuate dal solver, come $\mathscr{R} = \{R_i\}_{i=1}^{|\mathscr{R}|}$, dove R_i indica l'i-esima entità individuata dal solver. Le metriche qui proposte assumono che ciascuna entità sia distinta, ovvero che non esistano due entità che abbiano una menzione in comune, ovvero che:

$$\forall K_i, K_j \in \mathscr{R} | i \neq j \quad \nexists m | m \in K_i \wedge m \in K_j \tag{1.14a}$$

$$\forall R_i, R_j \in \mathscr{R} | i \neq j \quad \nexists m | m \in R_i \wedge m \in R_j \tag{1.14b}$$

Usiamo infine $\mathscr{M}_k$ e $\mathscr{M}_r$ per indicare le menzioni esatte (*key*) e le menzioni individuate dal solver (response). Rispetto alle metriche seguenti verranno considerati validi sia l'utilizzo di *gold mention* ($\mathscr{M}_k = \mathscr{M}_r$), sia quello di menzioni derivate dal solver ($\mathscr{M}_k \neq \mathscr{M}_r$). Se le menzioni utilizzate non sono esatte, è necessario scegliere una strategia per la loro valutazione. È infatti possibile utilizzare una metrica binaria per valutare la correttezza di una menzione, cioè:

$$s(R_i, K_j) = 1 \Leftrightarrow R_i = K_j \tag{1.15}$$

È possibile in alternativa fornire una metrica il cui punteggio misuri la correttezza delle due menzioni a seconda della loro similitudine, per esempio misurando la porzione a loro comune.

In modo analogo a quanto visto per le menzioni, è possibile utilizzare metriche binarie, che considerino esatte solo due entità identiche, oppure utilizzare criteri più granulari che misurino il grado di esattezza di due menzioni diverse.

La creazione di metriche adatte alla misurazione delle performance può adottare due modalità differenti per aggregare i risultati in un documento. Una scelta possibile è quella di contare in ciascuna entità il numero di menzioni considerate esatte, ottenere valori totali a partire da quelli di ciascuna entità ed effettuare in questo modo una media a livello del documento, in un approccio detto *mention-weighted*. Una seconda possibilità, detta *entity-weighted*, è quella di valutare la correttezza per ciascuna entità tramite un valore compreso fra 0 ed 1, sommando e calcolando la media di tutti i valori in un secondo momento.

Procediamo ora ad una descrizione delle metriche più diffuse per la valutazione della *coreference*: MUC-F, B-Cubed, CEAF e BLANC.

MUC F-measure

MUC F-measure [58] fu proposta nell'ambito dei task di *coreference* delle conferenze MUC-6 e MUC-7. Tale metrica opera misurando il numero di *coreference links*, ovvero il numero minimo di archi in un grafo contenente le

menzioni capaci di collegare in modo coerente le catene di coreferenza. Per calcolare il *recall* del sistema è necessario innanzitutto partizionare ciascuna *key entity* K_i rispetto alle *response mention* appartenenti a $\mathscr{R}$:

$$\mathscr{P}(K_i, \mathscr{R}) = \{K_i \cap R_j | j \in [1, \ldots, |\mathscr{R}|]\} \bigcup_{m \in (K_i - \mathscr{R})} \{\{m\}\} \tag{1.16}$$

Dove con $(K_i - \mathscr{R})$ si intende l'insieme di menzioni presenti in K_i ma assenti da tutti gli insiemi contenuti in $\mathscr{R}$. In altre parole con $\mathscr{P}(K_i, \mathscr{R})$ si indica un insieme formato da tutte le intersezioni di K_i con le *respose entities* R_i e tutti gli insiemi singoletto formati da menzioni presenti in K_i ma assenti dagli insiemi di $\mathscr{R}$.

Una volta costruita tale partizione si noti che il numero di collegamenti necessari per l'ottenimento dell'entità K_i a partire dagli insiemi $\mathscr{P}(K_i, \mathscr{R})$ è data da $|\mathscr{P}(K_i, \mathscr{R})| - 1$. Il numero di collegamenti minimi per ottenere l'entità K_i è invece dato da $|K_i| - 1$. Il numero di collegamenti corretti in $\mathscr{R}$ è dunque calcolabile come:

$$N_c = \sum_{i=1}^{|\mathscr{K}|}((|K_i| - 1) - (|\mathscr{P}(K_i, \mathscr{R})| - 1)) = \tag{1.17a}$$

$$= N_c = \sum_{i=1}^{|\mathscr{K}|}(|K_i| - |\mathscr{P}(K_i, \mathscr{R})|) \tag{1.17b}$$

In questi termini è dunque possibile calcolare il *recall* del sistema, ovvero il rapporto fra i collegamenti corretti e i collegamenti delle *key entity*:

$$r = \frac{\sum_{i=1}^{|\mathscr{K}|}(|K_i| - |\mathscr{P}(K_i, \mathscr{R})|)}{\sum_{i=1}^{|\mathscr{K}|}(|K_i| - 1)} \tag{1.18}$$

Occorre ora procedere per definire la *precision* del sistema, ovvero il rapporto fra il numero di collegamenti corretti nell'output del sistema e il numero di collegamenti effettivamente costruiti da esso. Si richiede dapprima di

invertire i termini della partizione utilizzata in precedenza, ottenendo quindi:

$$\mathscr{P}(R_j, \mathscr{K}) = \{K_i \cap R_j | i \in [1, \ldots, |\mathscr{K}|]\} \bigcup_{m \in (R_j - \mathscr{K})} \{\{m\}\} \tag{1.19}$$

In modo analogo a quanto visto in precedenza, la *precision* è esprimibile come:

$$p = \frac{\sum_{j=1}^{|\mathscr{R}|}(|R_i| - |\mathscr{P}(R_i, \mathscr{K})|)}{\sum_{j=1}^{|\mathscr{R}|}(|R_i| - 1)} \tag{1.20}$$

Da queste quantità è possibile derivare infine la MUC F-Measure, definita come la media armonica fra *precision* e *recall*:

$$\text{MUC} = \frac{2pr}{p + r} \tag{1.21}$$

MUC-F risulta essere una metrica interessante e tuttora utilizzata. Essa presenta però una serie di problematiche:

- Se immaginiamo un sistema che restituisca solo entità composte da una singola menzione, mentre esse non sono realmente singoletti nelle *key entities*, MUC-F computa una *precision* pari a 0/0. Una tale situazione può creare problematiche e deve essere gestita nell'implementazione di MUC ponendo in tale caso la *precision* a 0;

- Ciascun collegamento errato che viene prodotto ha lo stesso peso: è però lecito chiedersi se sia indifferente la presenza di un collegamento erroneo fra due entità di grandi dimensioni o fra una più piccola ed una più grande;

- Il sistema non penalizza a sufficienza l'unione di più entità, in quanto un unione fra due entità altrimenti corrette produce solo un collegamento errato, la cui influenza sulla metrica in totale è poco rilevante.

Per queste ragioni dopo la formalizzazione di MUC venne proposta la metrica B-cubed, di cui discutiamo in seguito.

B-cubed F-measure

Rispetto a MUC, B-cubed F-measure [6] è una metrica *mention-based*, proposta per risolvere alcuni problemi di MUC.

Per calcolare il *recall* di B-cubed è necessario, come per MUC, partizionare una *key entity* K_i rispetto alle *response entities* $\mathscr{R}$:

$$\mathscr{B}(K_i, \mathscr{R}) = \{K_i \cap R_j | j \in [1, \ldots, \mathscr{R}]\} \tag{1.22}$$

In questo caso $B(K_i, \mathscr{R})$ è l'insieme di tutti i sottoinsiemi che intersecano K_i con le *response entities* di $\mathscr{R}$. $|K_i \cap R_j|$ è quindi il numero di menzioni comuni alle entità K_i e R_j. Se si considerano anche punteggi di match parziali per le *response mentions*, $|K_i \cap R_j|$ deve essere modificato per considerare un grado di comunanza fra menzioni di K_i ed R_j.

Il contributo di ciascuna *key mention* $m \in K_i \cap R_j$ è dato da:

$$r(m) = \frac{|K_i \cap R_j|}{|K_i|} \tag{1.23}$$

Il contributo totale di tutte le menzioni in $K_i \cap R_j$ è dunque:

$$r(i, j) = \sum_{m \in K_i \cap R_j} r_m = \frac{|K_i \cap R_j|^2}{|K_i|} \tag{1.24}$$

Il *recall* totale del documento è dunque dato da:

$$r = \sum_{i,j} \frac{|K_i \cap R_j|^2}{|K_i||\mathscr{M}_k|} \tag{1.25}$$

Dove si è utilizzata la somma di tutti gli $r(i, j)$ applicando però una normalizzazione attraverso la cardinalità dell'insieme $\mathscr{M}$ delle *key mentions*.

Il calcolo della *precision* è ottenuto (come visto in precedenza per MUC) invertendo il ruolo di *key entities* e *response entities* nella partizione:

$$\mathscr{B}(R_j, \mathscr{K}) = \{K_i \cap R_j | i \in [1, \ldots, \mathscr{K}]\} \tag{1.26}$$

In modo analogo a quanto visto in precedenza, esprimiamo il contributo alla *precision* di ciascuna menzione nella *response entity* R_j rispetto a K_i è dato da:

$$p(i,j) = \sum_{i,j} \frac{|K_i \cap R_j|^2}{|R_j|} \tag{1.27}$$

Per ottenere la *precision* totale è dunque sufficiente normalizzare utilizzando il numero di *response mention* $|\mathscr{M}_r|$:

$$p = \sum_{i,j} \frac{|K_i \cap R_j|^2}{|R_i||\mathscr{M}_r|} \tag{1.28}$$

La metrica B-cubed F-score è quindi data dalla media armonica di *recall* e *precision*:

$$\text{B-cubed} = \frac{2pr}{p+r} \tag{1.29}$$

È possibile dimostrare come se le *response entities* non hanno menzioni in comune, la *recall* e la *precision* siano comprese fra 0 ed 1. Tuttavia, poiché la metrica valuta un punteggio per ogni menzione, l'aggiunta di duplicati può comportare un aumento dei valori di *precision* e *recall* arbitrario. È quindi responsabilità di chi utilizza questa metrica evitare questo tipo di situazioni, in quanto non è possibile per uno *scorer* B-cubed riconoscere quali fra le menzioni duplicate esso debba considerare.

La metrica così definita risolve gran parte delle problematiche che erano emerse per quanto riguarda MUC. Tuttavia essa valuta in maniera problematica sistemi "*dummy*": un sistema che unifichi tutte le menzioni in una stessa entità riceve un *recall* pari a 1, mentre un sistema che tratti ciascuna menzione come un'entità separata avrà precisione massima. Per risolvere queste problematiche è stato proposto l'utilizzo della metrica CEAF, che passiamo a descrivere.

Constrained Entity-Aligned F-Measure

La metrica CEAF [39] fu proposta per risolvere i problemi di B-cubed riguardanti la presenza di menzioni duplicate. La radice di tale problema venne individuata nella possibilità per B-Cubed di valutare un'entità più volte sulle stesse menzioni. Per questo CEAF si basa su allineamento uno ad uno delle entità, in modo che solo le entità allineate possano contribuire al punteggio finale.

Si indicano con m ed M rispettivamente il minimo ed il massimo numero di entità:

$$m = \min\{|\mathscr{K}, \mathscr{R}|\} \tag{1.30a}$$

$$M = \max\{|\mathscr{K}, \mathscr{R}|\} \tag{1.30b}$$

E si definiscono gli insiemi $\mathscr{K}_m \subset \mathscr{K}$ e $\mathscr{R}_m \subset \mathscr{R}$ come i sottoinsiemi rispettivamente di *key* e *response entity* contenenti esattamente m elementi.

Definiamo dunque due insiemi $G(\mathscr{K}_m, \mathscr{R}_m)$ e G_m come:

$$G(\mathscr{K}_m, \mathscr{R}_m) = \{g : \mathscr{K}_m \mapsto \mathscr{R}_m\} \tag{1.31a}$$

$$G_m = \bigcup_{(\mathscr{K}_m, \mathscr{R}_m)} G((\mathscr{K}_m, \mathscr{R}_m)) \tag{1.31b}$$

Dove $\mapsto$ denota una funzione iniettiva dall'insieme $\mathscr{K}$ all'insieme $\mathscr{R}_m$. Viene quindi richiesto che:

$$\forall g \in G(\mathscr{K}_m, \mathscr{R}_m), \forall R_i, R_j \in \mathscr{K}_m \quad R_i \neq R_j \Rightarrow g(R_i) \neq g(R_j) \tag{1.32}$$

Sia $\phi(K, R)$ una misura della similarità fra una *key entity* K ed una *response entity* R. Supponiamo inoltre che tale metrica fornisca valori positivi e che la similarità fra entità sia additiva, ovvero che per ogni $g \in G_m$, la similarità totale $\Phi(g)$ per una funzione g sia la somma delle similarità fra le coppie allineate da g:

$$\Phi(g) = \sum_{K \in \mathscr{K}_m} \phi(K, g(k)) \tag{1.33}$$

Dato un documento d e le sue *key entity* $\mathscr{K}$ e *response entity* $\mathscr{R}$ è possibile individuare il miglior allineamento fra le due:

$$g^* = \arg\max_{g \in G_m} \Phi(g) \tag{1.34a}$$

$$= \arg\max_{g \in G_m} \sum_{K \in \mathscr{K}_m} \phi(K, g(k)) \tag{1.34b}$$

Se si indicano con $\mathscr{K}_m^*$ e $\mathscr{R}_m^* = g^*(\mathscr{K}_m^*)$ rispettivamente i sottoinsiemi di *key* e *response entities* dove si ottiene g^*, la massima similarità totale è:

$$\Phi(g^*) = \sum_{k \in \mathscr{K}_m^*} \phi(K, g^*(K)) \tag{1.35}$$

Viene dunque imposto che $K = \emptyset \vee R = \emptyset \Rightarrow \phi(K, R) = 0$, eliminando il problema di effettuare *mapping* da un'entità ad un'entità vuota, poiché il *mapping* uno ad uno che massimizzi $\phi(g)$ deve appartenere a G_m.

Si noti come sia possibile calcolare la *self-similarity* di entità $\phi(K, K)$ e $\phi(R, R)$ per ogni $K \in \mathscr{K}$, $R \in \mathscr{R}$ utilizzando una semplice identità come mappa.

È ora possibile definire *recall*, *precision* e CEAF:

$$r = \frac{\Phi(g^*)}{\sum_i \phi(K_i, K_i)} \tag{1.36a}$$

$$p = \frac{\Phi(g^*)}{\sum_i \phi(R_i, R_i)} \tag{1.36b}$$

$$\text{CEAF} = \frac{2pr}{p + r} \tag{1.36c}$$

La metrica così definita ha proprietà interessanti: essa infatti penalizza i casi limite visti in precedenza, in particolare con bassa *precision* per troppe entità, *recall* basso per numero di entità insufficiente.

Un altro aspetto interessante riguarda il caso in cui non vi sia alcuna similarità fra le entità K ed R. In tal caso non è necessario calcolare $\phi(K, R)$, in quanto tale valore sarà pari a 0. Nelle implementazioni di CEAF dunque

questi casi vengono individuati ed ignorati nel calcolo di *precision* e *recall*.

Non sarà sfuggita al lettore una problematica che emerge dalle definizioni precedenti: l'individuazione dell'allineamento ottimale g^*. La ricerca tramite un algoritmo *brute force* di tale *mapping* richiederebbe infatti una ricerca fra tutte le possibili mappe *one-to-one* tra sottoinsiemi di $\mathscr{K}$ di dimensione m e sottoinsiemi di $\mathscr{R}$ di dimensione m. Il numero di tali mappe è però pari a $G_m = \binom{M}{m} m!$, che renderebbe intrattabile il problema anche per documenti con relativamente poche *entity*.

Fortunatamente il problema dell'allineamento fra entità può essere risolto in termini di *matching* bipartito massimo, utilizzando l'algoritmo di Kuhn-Munkres [34, 46], che ha complessità $O(Mm^2 \log(m))$. In un grafo pesato, il problema del *matching* bipartito massimo consiste nell'individuazione di un insieme di archi che non abbiano nodi in comune (*matching*) tali che la somma dei pesi degli archi individuata sia massima. Per effettuare questa riduzione è sufficiente considerare un grafo composto dalle entità di $\mathscr{K}$ ed $\mathscr{R}$ e da archi (K, R) dove $K \in \mathscr{K} \wedge R \in \mathscr{R}$. Ciascun arco ha poi un peso $\phi(K, R)$, che garantisce la proprietà desiderata: trovare un *matching* bipartito massimo equivale ad individuare l'allineamento ottimale fra i due insiemi.

Fino ad ora non si è definita in modo esplicito la misura della similarità fra entità $\phi(K, R)$. Esistono infatti due possibilità per la scelta di tale metrica:

$$\phi_m(K, R) = |K \cap R| \tag{1.37a}$$

$$\phi_e(K, R) = \frac{2|K \cap R|}{|K| + |R|} \tag{1.37b}$$

che corrispondono rispettivamente al numero di menzioni comuni fra le due entità e alla F-measure fra le due entità. L'utilizzo di ϕ_m o di ϕ_e da origine a due metriche distinte: rispettivamente si ha *mention-based* CEAF (CEAF_m) e *entity-based* CEAF (CEAF_e).

BiLateral Assesment of Noun-Phrase Coreference

Nella metrica MUC vengono considerati solamente i collegamenti fra menzioni di una stessa entità, per cui risulta difficoltoso la valutazione di entità singoletto, composte da una sola menzione.

Per risolvere tale problematica è stata proposta la metrica BiLateral Assessment of Noun-Phrase Coreference (BLANC) [53], che operava però inizialmente un'assunzione fondamentale: veniva infatti richiesto che le *key mention* fossero identiche alle *response mention*. In questo modo si sarebbero esclusi tutti i sistemi nei quali le menzioni vengono individuate in modo automatico, che potremmo definire *end-to-end*. Gli stessi autori hanno Fortunatamente eliminato questa restrizione in una estensione successiva della metrica [41], che sarà da noi esaminata.

A differenza di MUC, BLANC considera, oltre ai collegamenti fra menzioni di una stessa entità, anche link fra menzioni di entità distinte. Procediamo dunque alla definizione di *coreference links* formati da menzioni in K_i e R_j:

$$C_k(i) = \{(m_a, m_b) | m_a \in K_i \wedge m_b \in K_i \wedge m_a \neq m_b\} \tag{1.38a}$$

$$C_r(j) = \{(m_a, m_b) | m_a \in R_j \wedge m_b \in R_j \wedge m_a \neq m_b\} \tag{1.38b}$$

Questo tipo di collegamenti rappresentano una relazione di coreferenza all'interno di una entità, mentre per entità composte da una sola menzione si ha un insieme vuoto.

In modo duale, è necessario definire dei collegamenti fra menzioni detti di *non-coreference links*:

$$N_k(i, j) = \{(m_a, m_b) | m_a \in K_i \wedge m_b \in K_j\} \tag{1.39a}$$

$$N_r(i, j) = \{(m_a, m_b) | m_a \in R_i \wedge m_b \in R_j\} \tag{1.39b}$$

Tali insiemi sono formati da collegamenti fra menzioni non coreferenti nelle due entità K ed R, menzioni che dunque appartengono a entità differenti. Si noti come gli insiemi rispettino questa definizione solo se $i \neq j$, risultando

essere altrimenti equivalenti al prodotto delle parti di un insieme per sé stesso. Un altro aspetto interessante è il fatto che l'insieme di *non-coreference links* possa essere vuoto nel caso in cui tutte le menzioni facciano parte della stessa entità.

Definiamo ora, a partire dalle definizioni precedenti, le loro unioni come:

$$C_k^U = \bigcup_i C_k(i) \quad C_r^U = \bigcup_i C_r(i) \tag{1.40a}$$

$$N_k^U = \bigcup_{i \neq j} N_k(i,j) \quad N_r^U = \bigcup_{i \neq j} N_r(i,j) \tag{1.40b}$$

$$T_k = C_k^U \cup N_k^U \quad T_r = C_R^U \cup N_r^U \tag{1.40c}$$

È evidente come assumendo che le *key mention* e le *response mention* siano le stesse, si ha che $T_k = T_r$.

La definizione di BLANC fu adattata, utilizzando gli insiemi di collegamenti discussi in precedenza, a partire dalla RandIndex, una metrica per il *clustering*. Se *key mention* e *response mention* sono le stesse, è possibile applicare RandIndex in modo diretto, in quanto siamo in presenza di un problema di *clustering* tradizionale:

$$\text{RI} = \frac{|C_k^U \cap C_r^U| + |N_k^U \cap N_R^U|}{\frac{1}{2}(|T_k|(|T_k|-1))} \tag{1.41}$$

Tale formalizzazione, tuttavia, non tiene conto del fatto che il numero di *non-coreference links* è spesso in pratica molto maggiore dei *coreference links*. Tale disparità renderebbe la metrica poco sensibile a cambiamenti dell'insieme dei *coreference links*. Per tale ragione, BLANC opera calcolando una media degli F-score per *coreference links* e *non-coreference links*. Il *recall*, la *precision* e l'*F-score* dei *coreference links* sono quindi calcolati rispettivamente come:

$$R_c^{(g)} = \frac{|C_k^U \cap C_r^U|}{|C_k^U \cap C_r^U| + |C_k^U \cap N_r^U|} \tag{1.42a}$$

$$P_c^{(g)} = \frac{|C_k^U \cap C_r^U|}{|C_k^U \cap C_r^U| + |C_r^U \cap N_k^U|} \tag{1.42b}$$

$$F_c^{(g)} = \frac{2R_c^{(g)} P_c^{(g)}}{R_c^{(g)} + P_c^{(g)}} \tag{1.42c}$$

Vengono poi calcolati in modo analogo i valori di *recall*, *precision* e *F-score* per i *coreference links*:

$$R_n^{(g)} = \frac{|N_k^U \cap N_r^U|}{|N_k^U \cap C_r^U| + |N_k^U \cap N_r^U|} \tag{1.43a}$$

$$P_n^{(g)} = \frac{|N_k^U \cap N_r^U|}{|N_r^U \cap C_k^U| + |N_r^U \cap N_k^U|} \tag{1.43b}$$

$$F_n^{(g)} = \frac{2R_n^{(g)} P_n^{(g)}}{R_n^{(g)} + P_n^{(g)}} \tag{1.43c}$$

La metrica BLANC viene infine calcolata come media dei due *F-score*:

$$BLANC^{(g)} = \frac{F_c(g) + F_n(g)}{2} \tag{1.44}$$

Abbiamo dunque fornito la definizione di $BLANC^{(g)}$, che, come viene evidenziato dall'apice g è però in grado di operare solo su menzioni *golden standard*, che quindi sono uguali fra *key* e *response*. Si noti inoltre come l'utilizzo di una media aritmetica risolva il problema nella differenza di cardinalità elevata fra *coreference* e *non-coreference links*, in quanto essi ottengono lo stesso peso tramite la definizione 1.44.

Per poter utilizzare BLANC anche con menzioni non esatte, è necessario notare come:

- $C_k^U - T_r$ è l'insieme delle *key coreference links* della *key* non presenti nella *response*;
- $N_k - T_r$ è l'insieme delle *key non-coreference links* della *key* non presenti nella *response*;
- $C_k^U - T_r$ è l'insieme dei *coreference link* della *response* non presenti nella *key*;

- $N_k^U - T_r$ è l'insieme dei *non-coreference link* nella *response* non presenti nella *key*.

Possiamo dunque estendere la definizione di *recall*, *precision* e *F-measure* per i *coreference link* come segue:

$$R_c = \frac{|C_k^U \cap C_r^U|}{|C_k^U \cap C_r^U| + |C_k^U \cap N_r^U| + |C_k^U - T_r|} \tag{1.45a}$$

$$P_c = \frac{|C_k^U \cap C_r^U|}{|C_k^U \cap C_r^U| + |C_r^U \cap N_k^U| + |C_r - T_k|} \tag{1.45b}$$

$$F_c = \frac{2R_cP_c}{R_c + P_c} \tag{1.45c}$$

Mentre per quanto riguarda i *non coreference link* otteniamo *recall*, *precision* ed *F-measure* da:

$$R_n = \frac{|N_k^U \cap N_r^U|}{|N_k^U \cap C_r^U| + |N_k^U \cap N_r^U| + |N_k - T_r|} \tag{1.46a}$$

$$P_n = \frac{|N_k^U \cap N_r^U|}{|N_r^U \cap C_k^U| + |N_r^U \cap N_k^U| + |N_r - T_k|} \tag{1.46b}$$

$$F_n = \frac{2R_nP_n}{R_n + P_n} \tag{1.46c}$$

Possiamo infine definire la metrica BLANC estesa come:

$$BLANC = \frac{F_c + F_n}{2} \tag{1.47}$$

Tale metrica può dunque essere utilizzata anche qualora si utilizzino menzioni rilevate in modo automatico e risulta inoltre ben condizionata anche per quel che riguarda i casi limite.

CoNLL 2012

In questa sezione forniremo la definizione della metrica utilizzata nel task di *coreference* di CoNLL 2012, che è quella più diffusa per la valutazione dei sistemi di *coreference* contemporanei.

Si è discusso come non vi sia una metrica che risulti superiore a tutte le altre da ogni punto di vista, esistono bensì metriche che misurano le perfomance con modalità, operando tutte in modo valido e ragionevole. La metrica usata da CoNLL 2012 si basa pertanto su una media aritmetica di MUC, B-cubed e CEAF_e:

$$\frac{\text{MUC} + \text{B-cubed} + \text{CEAF}_e}{3} \tag{1.48}$$

1.8 Applicazioni di coreference resolution

Sebbene si siano discussi i principali aspetti riguardanti la *coreference resolution*, è interessante chiedersi se un tale solver possa avere un ruolo nell'ambito dell'elaborazione del linguaggio naturale più in generale oppure se la si tratti di una problematica di interesse prettamente linguistico.

Fra gli l'utilizzi di sistemi di *coreference* vi è indubbiamente la possibilità che venga utilizzato in sistemi di analisi semantica del testo: categorizzando le menzioni del testo in entità è possibile la costruzione automatica di un'ontologia che catturi le relazioni fra esse espresse nel documento. L'utilizzo di anafore di vario tipo (eg pronominali), infatti, rende interessante la possibilità di collegarle tramite *coreference chain* ad un'entità ben definita, idealmente rimuovendo ogni ambiguità.

Un'altra importante applicazione possibile, che può a sua volta utilizzare strumenti di analisi semantica, è quella dell'*automatic summary*, che ha lo scopo di effettuare un riassunto di un documento in modo automatico. Se immaginiamo un sistema che selezioni le frasi più rilevanti di un testo, è evidente come in presenza di anafore esso debba esplicitare le entità coinvolte, al fine di poter omettere frasi che esplicitano l'entità coinvolta ma che risultano essere meno significative. Un sistema di questo tipo, se sufficientemente avanzato, potrebbe persino creare catene di anafore proprie a partire da quelle già presenti nel testo.

Un ulteriore aspetto è quello legato alla traduzione automatica fra lin-

gue diverse. In lingue che prevedano l'uso di *zero-pronoun*, un sistema di traduzione potrebbe utilizzare un sistema di *coreference* per rilevare questo tipo di pronomi e capire a quale entità essi si riferiscano. Se il pronome non è esplicitato ma è richiesto nella lingua che si intende ottenere, è necessario comprendere a quale entità ci si riferisca per scegliere il pronome di genere appropriato. In modo analogo, se si traduce da una lingua che preveda genere sintattico ad una con genere semantico o viceversa, è necessario comprendere l'entità a cui appartengono i pronomi, in modo da utilizzare il genere appropriato.

Si sono dunque elencate alcune possibili applicazioni di un solver per la *coreference* in ambiti più ampi, mostrando come l'interesse per la risoluzione del problema specifico possa introdurre benefici per altri task di elaborazione del linguaggio naturale. Avendo descritto completamente il fenomeno in esame e di alcune proposte per la sua risoluzione automatica, discuteremo ora delle reti neurali, della loro applicazione a task di elaborazione del linguaggio naturale e dei sistemi di *coreference resolution* basati su di esse, in modo da chiarire le ragioni del loro utilizzo nel nostro solver.

Capitolo 2

Reti Neurali ed elaborazione del linguaggio naturale

Negli ultimi anni si è assistito ad un riemergere delle reti neurali nell'ambito del *machine learning*, soprattutto per problemi di intelligenza artificiale. Le reti neurali e in particolare la *backpropagation* [59], lo strumento principale utilizzato per effettuare apprendimento supervisionato, furono sviluppati già negli anni 70, tuttavia la possibilità di utilizzare reti di dimensioni adeguate a risolvere problemi complessi è stata fino a pochi anni fa limitata dalla potenza di calcolo dell'hardware.

In questo capitolo forniremo dapprima una descrizione delle reti neurali e delle loro principali componenti, in seguito ci si concentrerà su strumenti e problematiche legate all'utilizzo delle reti neurali nell'ambito dell'elaborazione del linguaggio naturale.

2.1 Reti neurali

In questa sezione forniremo una definizione formale di rete neurale, a cui seguirà una descrizione di tutte le componenti principali di tale tecnologia. Verranno inoltre mostrate alcune innovazioni rispetto al modello tradizionale

di rete neurale che hanno condotto a miglioramenti sostanziali in branche disparate dell'informatica.

Il vantaggio principale delle reti neurali, secondo l'opinione dell'autore, è la loro versatilità: si tratta di modelli che, pur non essendo completamente *domain-independent*, forniscono strumenti standard che, combinati in maniera adeguata, possono essere utilizzati per scopi molto diversi.

2.1.1 Machine Learning

Le reti neurali rientrano in una categoria di algoritmi detti di *machine learning*. Tali algoritmi sono a loro volta suddivisi a seconda della modalità di apprendimento che essi sfruttano. Si hanno infatti tre categorie principali di apprendimento [54]:

- **Supervised Learning**: all'algoritmo vengono forniti dei dati annotati, costituiti da esempi di input ed esempi di output, dai quali l'algoritmo apprende come risolvere il problema;
- **Unsupervised Learning**: all'algoritmo non vengono fornite etichette di output. Esso deve apprendere in modo autonomo (non supervisionato) la struttura degli input che gli vengono forniti ed individuare al loro interno dei pattern. Un tale approccio può essere utilizzato per scoprire pattern ignoti nei dati o per risolvere uno specifico problema noto;
- **Reinforcement Learning**: all'algoritmo viene fornito un flusso di dati, sulla base dei quali esso compie delle scelte, a partire dalle quali ad esso viene fornito un feedback in termini di ricompensa o punizione.

In questo capitolo ci si concentrerà su reti neurali che operano una modalità di apprendimento supervisionato, che è il meccanismo utilizzato nel nostro *solver*, nonché l'utilizzo più comune delle reti neurali.

Gli algoritmi di apprendimento supervisionato operano dunque in due fasi distinte: una fase di *training*, nella quale gli vengono forniti i dati annotati in

modo che essi possano apprendere come risolvere un dato problema ed una fase di *testing* nella quale si valutano le performance degli algoritmi. Al fine di ottenere delle valutazioni realistiche, è pertanto necessario segmentare il *dataset* annotato in modo che il *testing* venga effettuato su dati distinti da quelli utilizzati per il *training*. La segmentazione del *dataset* risulta tipicamente nei seguenti sottoinsiemi distinti:

- **Training set**: sono i dati utilizzati in fase di training;
- **Validation set**: sono dati utilizzati per ottenere una valutazione intermedia delle performance dell'algoritmo in modo indipendente rispetto al *test set* ed individuare le situazioni di *overfitting* rispetto al *training set*;
- **Test set**: sono i dati utilizzati per il *testing* finale.

Formalmente, un algoritmo di apprendimento supervisionato opera su esempi annotati della forma $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ dove x_i, y_i rappresentano rispettivamente un dato e l'etichetta ad esso associato. Si tratta dunque di individuare una funzione $g : X \Rightarrow Y$ che mappa lo spazio degli input nello spazio delle soluzioni. Il meccanismo per individuare tale funzione dipende dalla tipologia di algoritmo utilizzata.

2.1.2 Definizione di rete feed-forward

Una rete neurale *feed-forward* è un modello relativamente semplice di rete neurale. Essa è formata da un insieme di nodi ed archi, attraverso cui viene propagato l'input fino a giungere a nodi di output che effettuano una classificazione di qualche tipo.

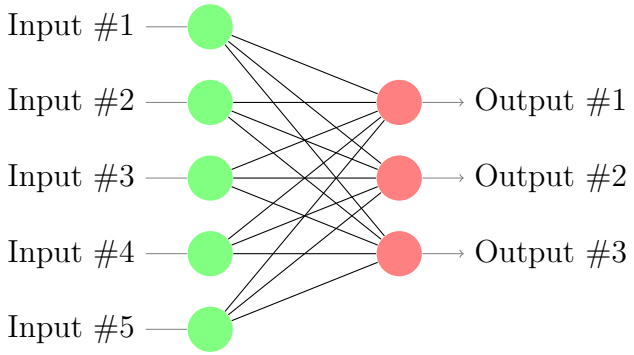


Figura 2.1: Una rete neurale single layer

Il modello più semplice di rete neurale *feed-forward* è una rete *single-layer* (figura 2.1), che è composta dunque da soli due livelli di nodi:

- Livello di input: comprende i nodi attraverso i quali l'input viene inserito nella rete, prima di essere propagato al suo interno. Il livello di input è costituito da un numero di nodi pari alla dimensione del vettore di input, che deve avere quindi una dimensione prestabilita;

- Livello di output: comprende uno o più nodi di output. Nell'ambito di problemi di categorizzazione, viene utilizzato un solo nodo se si richiede una scelta binaria, mentre si utilizza un numero di nodi pari alle categorie se esse sono più numerose.

I nodi di input e quelli di output di tale rete sono collegati da una serie di archi pesati, i cui pesi costituiscono i parametri della rete e che vengono modificati per risolvere il problema in esame. È necessario definire il vettore di input $\boldsymbol{I} \in \mathbb{R}^{d_{in}}$, il vettore degli output $\boldsymbol{O} \in R^{d_{in}}$ e i pesi associati a ciascun arco fra i nodi di input ed output, rappresentati da una matrice $\boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_{out}}$, nonché un vettore di bias $\boldsymbol{b} \in \mathbb{R}^{d_{out}}$. È dunque esprimere una relazione fra gli input e ciascuno specifico output come:

$$o_j = f(\sum_{i=0}^{ni} w_{i,j} + b_i) \quad \forall j \in [0, \ldots, d_{out}] \tag{2.1}$$

Dove f è una funzione detta di attivazione, della cui natura discuteremo in seguito. È evidente come i valori di output o_j siano una semplice composizione lineare degli input con i pesi $w_{i,j}$. È dunque possibile esprimere il vettore di output in forma algebrica:

$$\boldsymbol{O} = g(\boldsymbol{IW} + \boldsymbol{b}) \tag{2.2}$$

Dove g è una funzione che applica f su ciascun elemento del vettore $\boldsymbol{IW} + \boldsymbol{b}$:

$$g(\boldsymbol{X}) = (f(x_1), \ldots, f(x_{d_{out}})) \tag{2.3}$$

Tale tipologia di rete, sebbene sia in grado di risolvere problemi di natura semplice, può essere resa più espressiva aggiungendo uno o più livelli di nodi detti nascosti.

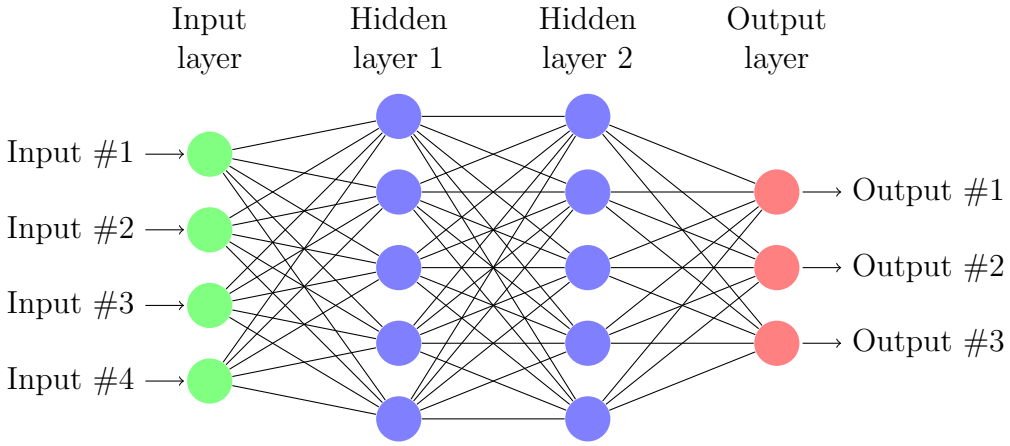


Figura 2.2: Rete neurale con due livelli nascosti

Questa tipologia di rete è strutturata in modo analogo a quanto visto in precedenza, ma sfrutta uno o più livelli aggiuntivi (detti nascosti).

Formalmente, utilizzando una rete con due livelli nascosti (figura 2.2) di dimensioni $h1, h2$ rispettivamente, si definiscono tre matrici di pesi $\boldsymbol{W}_{h1} \in \mathbb{R}^{d_{in} \times d_{h1}}, \boldsymbol{W}_{h2} \in \mathbb{R}^{d_{h1} \times d_{h2}}, \boldsymbol{W}_{out} \in \mathbb{R}^{d_{h2} \times d_{out}}$ e vettori di bias $\boldsymbol{b}_{h1} \in \mathbb{R}^{d_{h1}}, \boldsymbol{b}_{h2} \in \mathbb{R}^{d_{h2}}, \boldsymbol{b}_{out} \in \mathbb{R}_{d_{out}}$.

L'output del primo livello nascosto sarà dunque dato da:

$$\boldsymbol{H}_1 = g_1(\boldsymbol{I}\boldsymbol{W}_{h1} + \boldsymbol{b}_{h1}) \tag{2.4}$$

Il secondo livello nascosto sarà dunque:

$$\boldsymbol{H}_2 = g_2(\boldsymbol{H}_1\boldsymbol{W}_{h2} + \boldsymbol{b}_{h2}) \tag{2.5}$$

Esplicitando le due formule precedenti otterremo la formula per ottenere il vettore di output:

$$\begin{aligned} \boldsymbol{O} &= g_{out}(\boldsymbol{H}_2\boldsymbol{W}_{out} + \boldsymbol{b}_{out}) && \text{(2.6a)} \\ &= g_{out}(g_2(\boldsymbol{H}_1\boldsymbol{W}_{h2} + \boldsymbol{b}_{h2})\boldsymbol{W}_{out} + \boldsymbol{b}_{out}) && \text{(2.6b)} \\ &= g_{out}(g_2(g_1(\boldsymbol{I}\boldsymbol{W}_{h1} + \boldsymbol{b}_{h1})\boldsymbol{W}_{h2} + \boldsymbol{b}_{h2})\boldsymbol{W}_{out} + \boldsymbol{b}_{out}) && \text{(2.6c)} \end{aligned}$$

Dove g_1, g_2, g_{out} sono le funzioni di attivazione per i rispettivi livelli della rete.

È evidente come tale approccio sia scalabile per un numero di livelli arbitrario. Questo approccio, che coinvolge reti neurali con molti livelli nascosti, viene comunemente definito *deep learning*. L'utilizzo di funzioni di attivazione non lineari e di uno stack di livelli nascosti sufficientemente profondo consente infatti di apprendere rappresentazioni intermedie del problema via via più sofisticate, che consentono di affrontare problemi più complessi.

2.1.3 Funzioni di attivazione

Si è fornita in precedenza una breve descrizione della struttura di base delle reti neurali *feed forward*, mentre si è solo descritto in breve il ruolo delle funzioni di attivazione associate a ciascun livello della rete. Per la scelta di tale funzioni esistono varie alternative, che hanno caratteristiche diverse ma che soddisfano un requisito comune: sono facilmente derivabili. Fra tali alternative la più basilare è quella di scegliere una funzione di attivazione lineare.

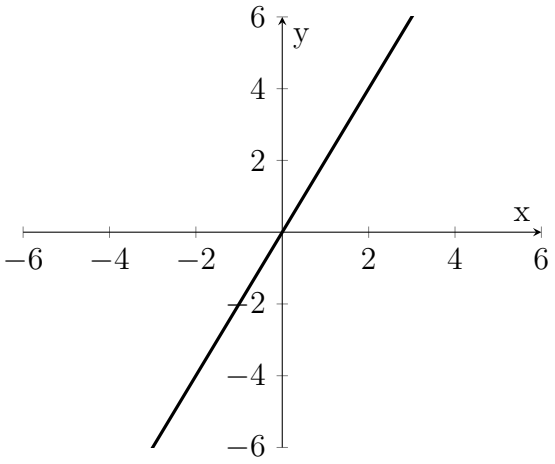


Figura 2.3: Grafico della funzione lineare

$$f(x) = cx, c \in \mathbb{R} \tag{2.7}$$

Tali funzioni risultano essere molto semplici da calcolare, derivabili in modo elementare (la loro derivata è la costante c), ma sono lineari. Le funzioni di attivazione applicate ai livelli intermedi di una rete vengono utilizzate anche per introdurre non linearità nella rete stessa, in modo da aumentarne la potenza espressiva. Per questa ragione un'attivazione lineare è scarsamente utilizzata, mentre esistono altre proposte che sono più comuni, quali ad esempio la funzione logistica:

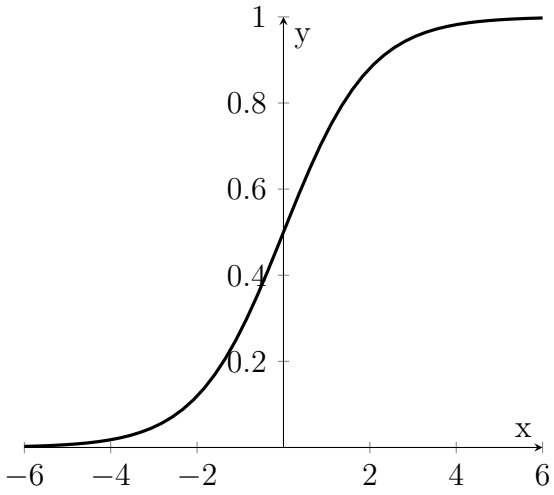


Figura 2.4: Grafico della funzione logistica

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

È evidente come tale funzione non sia lineare ed abbia inoltre l'interessante caratteristica di avere codominio compreso fra 0 ed 1. La funzione è inoltre derivabile utilizzando la *chain rule*:

$$\begin{aligned}
\frac{\partial}{\partial x}\sigma(x) &= \frac{\partial}{\partial x}(1 + e^{x})^{-1} && \text{(2.9a)}\\
&= (1 + e^{x})^{-2}\frac{\partial}{\partial x}(1 + e^{x}) && \text{(2.9b)}\\
&= \frac{1}{(1 + e^{x})^{2}}\, e^{-x}(-1) && \text{(2.9c)}\\
&= \frac{1}{1 + e^{x}}\frac{1}{1 + e^{x}}(-e^{-x}) && \text{(2.9d)}\\
&= \frac{1}{1 + e^{x}}\frac{-e^{-x}}{1 + e^{-x}} && \text{(2.9e)}\\
&= \frac{e^{-x}}{(1 + e^{-x})^{2}} && \text{(2.9f)}\\
&= \frac{1 + e^{-x}}{(1 + e^{-x})^{2}} - \frac{1}{(1 + e^{-x})^{2}} && \text{(2.9g)}\\
&= \frac{1}{1 + e^{-x}} - \left(\frac{1}{(1 + e^{-x})}\right)^{2} && \text{(2.9h)}
\end{aligned}$$

$$= \sigma(x) - \sigma(x)^2 \tag{2.9i}$$

Tale derivata ha quindi il grande vantaggio di poter essere espressa in termini di $\sigma(x)$, che può facilmente essere memorizzata durante l'attivazione della rete.

Fra le funzioni utilizzabili come funzioni di attivazione vi è poi la tangente iperbolica, che ha caratteristiche simili a quelle della funzione logistica.

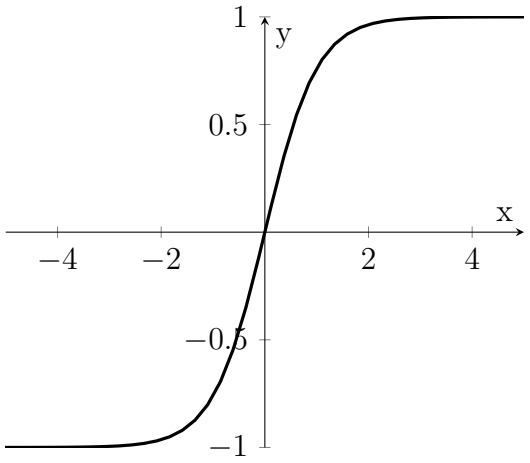


Figura 2.5: Grafico della tangente iperbolica

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} \tag{2.10}$$

Tale funzione, come nel caso di $\sigma(x)$, ha un codominio limitato, in questo caso compreso fra -1 ed 1. Come la funzione logistica, inoltre, ha una forma paragonabile ad una S e non è lineare. Il calcolo della sua derivata risulta inoltre semplificabile tramite *chain rule* e regole di derivazione di seno e coseno iperbolici:

$$\frac{\partial}{\partial x}\tanh(x) = \frac{\partial}{\partial x}\frac{\sinh(x)}{\cosh(x)} \tag{2.11a}$$

$$= \frac{\cosh(x)\frac{\partial}{\partial x}\sinh(x) - \sinh(x)\frac{\partial}{\partial x}\cosh(x)}{\cos^2(x)} \tag{2.11b}$$

$$= \frac{\cosh(x)\cosh(x) - \sinh(x)\sinh(x)}{\cos^2(x)} \tag{2.11c}$$

$$= \frac{\cosh^2(x) - \sinh^2(x)}{\cos^2(x)} \tag{2.11d}$$

$$= 1 - \frac{\sinh^2(x)}{\cosh^2(x)} \tag{2.11e}$$

$$= 1 - \tanh^2(x) \tag{2.11f}$$

Come nel caso della funzione logistica, è dunque possibile valutare la derivata di $tanh(x)$ in un punto a partire dal valore della funzione stessa, che è già stato calcolato durante l'attivazione della rete e può quindi essere mantenuto in memoria.

Un ultima funzione di attivazione diffusa nell'ambito delle reti neurali è detta Rectifier Linear Unit (ReLU) [24]:

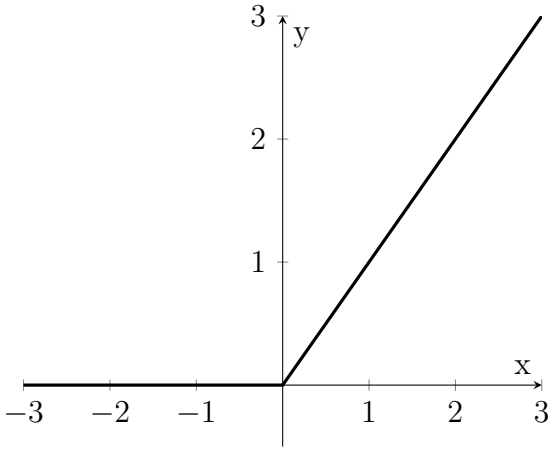


Figura 2.6: Grafico della funzione ReLU

$$\text{ReLU}(x) = \max(0, x) = \begin{cases} 0 & \text{se } x < 0 \\ x & \text{altrimenti} \end{cases} \tag{2.12}$$

Tale funzione si limita dunque a porre a 0 tutti i valori negativi di x. Nonostante si tratti di una funzione molto semplice da calcolare, essa ha dimostra-

to nelle applicazioni pratiche di poter produrre ottimi risultati, generalmente migliori anche delle altre funzioni qui presentate.

La sua derivata prima è calcolabile in modo elementare come:

$$\frac{\partial}{\partial x}\text{ReLU}(x) = \begin{cases} 1 & \text{se } x > 0 \\ 0 & \text{se } x < 0 \end{cases} \tag{2.13}$$

La funzione tuttavia presenta un punto di discontinuità per $x = 0$ e il valore di $\frac{\partial}{\partial x}\text{ReLU}(x)$ è tecnicamente indefinito. Nella maggior parte delle implementazioni si pone dunque $\frac{\partial}{\partial x}\text{ReLU}(x) = 0$ in modo arbitrario, risolvendo tale problematica.

Finora si è discusso di funzioni di attivazione che vengono generalmente applicate ai livelli intermedi di una rete neurale. È tuttavia comune l'utilizzo di funzioni di attivazione anche per quel che riguarda il livello di output della rete. Nel caso in cui si utilizzi un solo nodo di output, si utilizza solitamente una funzione logistica per normalizzare i valori fra 0 ed 1, in modo da poterli interpretare come valori binari. Se si intendono utilizzare multiple categorie, invece, è necessario utilizzare un'altra funzione di attivazione, chiamata softmax. A differenza delle funzioni fin qui esaminate, softmax non opera su singoli elementi di un vettore bensì sull'intero vettore, ed è definita come:

$$S(\boldsymbol{x}) = \frac{e^{x_i}}{\sum_{j=0}^{n} e^{x_k}} \tag{2.14}$$

Dove $\boldsymbol{x} \in \mathbb{R}^n$ e $n \in \mathbb{N}$. Tale funzione ha la seguenti proprietà:

$$\sum_{i=0}^{n} o_i = 1 \wedge \forall i \in [0, \ldots, n]\ \boldsymbol{o}_i \geq 0 \tag{2.15}$$

Dove $\boldsymbol{o} = S(\boldsymbol{x})$ è un vettore ottenuto tramite l'applicazione di softmax a un vettore $\boldsymbol{x}$. Tali proprietà risultano essere le stesse di una distribuzione di probabilità. Se individuiamo con $C = [1, \ldots, n]$ il numero di classi in cui categorizzare l'input, è possibile intepretare una rete al cui livello di output

viene applicato softmax come:

$$o_i = P(\boldsymbol{I} \in i) \tag{2.16}$$

Ovvero come la probabilità che gli input appartengano alla i-esima categoria.

Rispetto alle funzioni precedenti, che operano su elementi singoli di un vettore in una rete neurale, softmax mappa il vettore a un altro vettore di dimensione identica. In questo senso, tale funzione non ammette una derivata prima in senso stretto, bensì una matrice jacobiana, ovvero una matrice di derivate parziali nelle sue variabili (gli elementi del vettore). Formalmente la matrice jacobiana di una funzione $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ è definita come:

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_n} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{2.17}$$

Dove per $\frac{\partial f_i}{\partial x_j}$ si intende la derivata dell'i-esimo output rispetto al j-esimo input.

Trattandosi di funzioni applicate a vettori, anche per quanto riguarda le altre funzioni il calcolo delle derivate parziali utilizzato per l'algoritmo di *backtracking* calcola in sostanza una matrice jacobiana, che non è però stata esplicitata nei casi precedenti in quanto si trattava di funzioni *element-wise* che non richiedono una jacobiana per definire la propria derivata.

Per quanto riguarda softmax, essa è una funzione del tipo $S : \mathbb{R}^n \mapsto \mathbb{R}^n$ che ha quindi una matrice jacobiana di dimensione $n \times n$. È possibile dimostrare che gli elementi della matrice jacobiana $J_S = \frac{\partial f_i}{\partial x_j}$ applicata a sono esprimibili nella forma:

$$\boldsymbol{J}_{ij} = \begin{cases} f_j(1 - f_i) & \text{se } i \neq j \\ -f_j f_i & \text{altrimenti} \end{cases} \tag{2.18}$$

Tramite osservazioni sulla natura della matrice jacobiana ottenuta da una

funzione softmax è possibile calcolare tale derivata in maniera efficiente, senza ricorrere alla moltiplicazione di matrici di grandi dimensioni.

Si sono discusse le principali funzioni di attivazione comunemente utilizzate nell'ambito delle reti neurali. Prima di poter discutere la grande innovazione tecnica introdotta attraverso le reti neurali, ovvero l'algoritmo di *backpropagation*, è necessario introdurre il concetto di funzione costo o *loss*, che verrà trattato nella prossima sezione.

2.1.4 Funzione costo e categorical cross entropy

La funzione costo o loss è uno strumento indispensabile per l'implementazione dell'algoritmo di *backpropagation*, che è alla base dell'apprendimento supervisionato delle reti neurali. Come è intuibile dal nome stesso di tale funzione, esso fornisce una metrica dell'errore dell'algoritmo, che viene utilizzata dall'algoritmo di *backpropagation* per modificare i pesi della rete in modo che essi approssimino la funzione richiesta.

Fra le possibili funzioni *loss* vi è lo scarto quadratico medio, che viene definito come:

$$E = \frac{\sum_{i=1}^{n} (\hat{y}_i - y_i)^2}{n} \tag{2.19}$$

Dove:

- $x = (x_1, \ldots, x_n)$ è il vettore di input del training set;
- $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_n)$ sono le etichette associate a ciascun input;
- $y = (y_1, \ldots, y_n)$ è il vettore delle previsioni effettuate dalla rete sui rispettivi input.

Un'altra possibilità per la scelta di funzioni *loss* è la *cross-entropy*, che viene utilizzata in particolare quando la rete deve risolvere un problema di classificazione.

Nell'ambito della teoria dell'informazione, la *cross-entropy* fra due distribuzioni di probabilità p e q sullo stesso insieme di eventi misura il numero

medio di bit necessario per identificare (categorizzare) un evento dell'insieme, se si utilizza una codifica che è stata ottimizzata per una distribuzione di probabilità errata q, invece di quella reale p. Formalmente se si opera su distribuzioni di probabilità p e q discrete:

$$H(p, q) = -\sum_{x} p(x) \log q(x) \tag{2.20}$$

Per utilizzare tale valore come stima dell'errore è dunque sufficiente considerare una serie di dati annotati della forma $x_i, y_i \ \forall i \in [1, \ldots, n]$, che corrispondono rispettivamente agli input ed alle loro etichette ed una serie di stime $\hat{y}_i \ \forall i \in [1, \ldots, n]$ fornite dalla rete. È dunque possibile utilizzare come metrica della *loss* su un singolo input la *cross-category entropy* nel modo seguente:

$$L = -\sum_{i=0}^{n} (y_i \log \hat{y}_i) \tag{2.21}$$

Tale metrica è stata dunque adattata alla misurazione delle performance della nostra rete neurale. Se supponiamo che il problema in esame sia una categorizzazione esclusiva, nella quale solo una scelta è corretta t, la *cross-entropy* può essere semplificata nel modo seguente:

$$L = -\log(\hat{y}_t) \tag{2.22}$$

In questo caso si misura solo la performance della classe corretta, poiché essa è l'unica ad essere interessante per la risoluzione del problema.

Si è dunque conclusa la discussione del calcolo delle funzione costo tramite *categorical cross-entropy*.

2.1.5 Backpropagation e Gradient Descent

Il meccanismo alla base dell'utilizzo di reti neurali per l'apprendimento supervisionato è la *backpropagation* [59]. Una volta definita una rete neurale, vengono inizializzati i suoi pesi $\boldsymbol{W}_i$ ed i vettori di *bias* $\boldsymbol{b}_i$ per ciascun livel-

lo della rete $i \in [1, \ldots, nl]$ tramite un qualche algoritmo di generazione di numeri pseudocasuali. La *backpropagation* è dunque strutturato nel modo seguente:

1. Si calcola l'ouput della rete per un dato input x_j;
2. Viene calcolata una funzione costo (*loss*) dell'ouput così ottenuto rispetto all'etichetta $\hat{y}_j$ corrispondente all'input x_j;
3. Si applica un passo di ottimizzazione, che procede a ritroso sulla rete, modificando i pesi attraverso uno *step* predefinito, in modo da minimizzare l'errore sul dato in input;
4. Si ripete il processo finché non si verifica un criterio di interruzione, utilizzando più volte gli stessi dati di input in maniera ciclica se necessario.

Per utilizzare tale tecnica è dunque necessario definire un ottimizzatore, che opera sui pesi della rete in modo da migliorarne le perfomance. Alla base della maggior parte degli ottimizzatori utilizzati nell'ambito delle reti neurali è l'utilizzo di passi di apprendimento proporzionali al gradiente negativo della funzione costo. L'algoritmo di ottimizzazione che utilizza questa tecnica è detto *Stochastic Gradient Descent* [37] e sfrutta le informazioni sul gradiente della funzione di *loss*, calcolata dalla rete neurale, per determinare in che direzione debbano muoversi i pesi per avvicinarsi ad un minimo della funzione costo.

Essendo un algoritmo di ottimizzazione generico, esso opera su una funzione generica $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ formata da parametri θ alla ricerca di parametri che minimizzino l'errore sui dati $\boldsymbol{I} = (x_i, y_i)$

Algoritmo 2 Stochastic Gradient Descent

1: **Input**: $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ parametrized by θ
2: **Input**: training set $\boldsymbol{I} = (x_i, y_i), i \in [1, \ldots, d_s]$
3: **Input**: learning rate η

4: **Input**: loss function L
5: $i = 1$
6: **while** the halting conditions aren't met **do**
7: $\quad \hat{g} \leftarrow \nabla(L(f(x_i), y_i)$
8: $\quad \theta \leftarrow \theta + \eta\hat{g}$
9: $\quad i = (i + 1) \mod d_{in}$
10: **end while**

Tale tipologia di algoritmo sfrutta dunque il gradiente (più propriamente la sua estensione per funzioni vettoriali, la matrice jacobiana) per determinare in che modo sia necessario modificare i parametri θ della funzione f. Forniamo ora un breve esempio di applicazione di tale algoritmo ad una rete neurale con un solo livello nascosto.

Sia NN una rete neurale con un solo livello nascosto, pesi associati ai livelli della rete dati da $\boldsymbol{W}_1 \in \mathbb{R}^{2\times2}, \boldsymbol{W}_2 \in \mathbb{R}^{2\times2}$, funzioni di attivazione dei due livelli $f_1(x) = f_2(x) = x$ e sia $(x_i, y_i), x_i \in \mathbb{R}^2 \wedge y_i \in \mathbb{R}^2$ un esempio annotato per tale rete. Ricordiamo dunque come l'output di un determinato neurone di output h_i sia dato da:

$$h_i = \sigma(u_i) = \sigma(\sum_{j=0}^{2}(w_{ji}x_j)) \tag{2.23}$$

Analogamente, l'output di ciascun neurone di output o_i è dato da:

$$o_i = \sigma(u'_i) = \sigma(\sum_{j=0}^{2}(w'_{ij}h_j)) \tag{2.24}$$

Dove w_{ij} sono i pesi della matrice $\boldsymbol{W}_2$. La funzione *loss*, espressa tramite scarto quadratico medio è dunque:

$$L = \frac{1}{2}\sum_{i=0} 2(o_i - y_i)^2 \tag{2.25}$$

Per calcolare il gradiente della funzione *loss* rispetto ai pesi associati agli

archi compresi fra il livello nascosto e quello di output si applica la *chain rule*:

$$\frac{\partial L}{\partial w'_{ij}} = \frac{\partial L}{\partial o_j}\frac{\partial o_j}{\partial u'_j}\frac{\partial u'_j}{\partial w_{ij}} \tag{2.26}$$

La derivata di L rispetto a o_j è data da:

$$\frac{\partial L}{\partial o_j} = o_j - y_j \tag{2.27}$$

Dalla sezione precedente si ricordi come la derivata della funzione σ rispetto a u'_j sia data da $\sigma(x) - \sigma^2(x)$. Sostituendo si ottiene:

$$\frac{\partial o_j}{\partial u'_j} = o_j - o_j^2 \tag{2.28}$$

Si procede infine al calcolo della derivata di $u' = \sum_{i=0}^{n} w'_i j h_i$ rispetto a w_{ij}, che corrisponde a h_i. Applicando le sostituzioni all'equazione 2.26, si ottiene:

$$\frac{\partial L}{\partial w'_{ij}} = (o_j - y_j)(o_j - o_j^2)h_i \tag{2.29}$$

Che corrisponde al gradiente per il peso w'_{ij}, che possiamo utilizzare per ricavare la regola di aggiornamento per tale peso:

$$w'^{new}_{ij} = w'^{old}_{ij} - \eta((o_j - y_j)(o_j - o_j^2)h_i) \tag{2.30}$$

Si consideri ora la derivata della funzione obiettivo rispetto ad un peso di un arco fra livello di input e livello nascosto w_{ij}. In modo analogo a quanto visto in precedenza, si ottiene la formula:

$$\frac{\partial L}{\partial w_{ji}} = \sum_{k=0}^{2}\left(\frac{\partial L}{\partial o_l}\frac{\partial o_k}{\partial u'_k}\frac{\partial u'_k}{\partial h_i}\right)\frac{\partial h_i}{\partial u_i}\frac{\partial u_i}{\partial w_{ji}} \tag{2.31}$$

Si noti come $\frac{\partial L}{\partial o_j}$ e $\frac{\partial o_j}{\partial u'_j}$ siano già stati calcolati nei passaggi precedenti (equazioni 2.27 e 2.28). Si procede dunque calcolando le derivate rimanenti:

$$\frac{\partial u'_j}{\partial h_i} = w'_{ij} \tag{2.32a}$$

$$\frac{\partial h_i}{\partial u'_i} = h_i - h_i^2 \tag{2.32b}$$

$$\frac{\partial u_i}{\partial w_{ji}} = x_k \tag{2.32c}$$

Sostituendo in 2.31 si ottiene:

$$\frac{\partial L}{\partial w_{ji}} = \sum_{k=1} 2((o_j - y_j)(o_j - o_j^2)w'_{ij})(h_i - h_i^2)x_k \tag{2.33}$$

Dalla quale si deriva la regola di aggiornamento per w_{ij}

$$w_{ij}^{new} = w_{ij}^{old} - \eta \sum_{k=1} 2((o_j - y_j)(o_j - o_j^2)w'_{ij})(h_i - h_i^2)x_k \tag{2.34}$$

Si noti come per ottenere le derivate parziali di ciascun peso si utilizzino derivate parziali già calcolate nei passaggi precedenti. Tale struttura consente una formalizzazione delle reti neurali che utilizza un grafo computazionale: la rete viene rappresentata da un grafo, che è viene attivato in avanti per valutare gli input, mentre viene percorso a ritroso per calcolare le derivate parziali necessarie al calcolo dei gradienti.

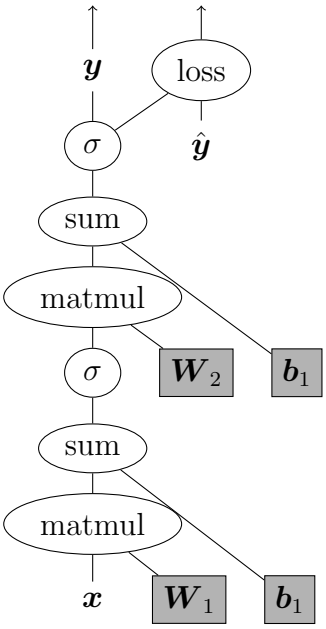


Figura 2.7: Grafo computazionale di una rete neurale

Viene mostrata in figura 2.7 la struttura di una semplice rete neurale con un livello nascosto. Si noti come sia possibile percorrendo il grafo dagli input $\boldsymbol{x}$ all'output $\boldsymbol{y}$ oppure ottenere la *loss* L dall'output della rete e dalle etichette $\hat{\boldsymbol{y}}_i$. Percorrendo a ritroso la rete dal ramo relativo alla *loss* è possibile applicare la *backpropagation*, ad esempio tramite *stochastic gradient descent*.

È evidente come tale struttura consenta di costruire reti con strutture molto complesse, senza tuttavia dover operare modifiche all'algoritmo di ottimizzazione. È questa la ragione per cui esistono librerie per il *deep learning* che consentono una libertà di implementazione totale mediante la semplice definizione della rete e dei suoi parametri.

Si è dunque conclusa la descrizione del funzionamento dell'apprendimento tramite *backpropagation* e *stochastic gradient descent*. Esistono però algoritmi di apprendimento più avanzati, dei quali discutiamo in seguito.

2.1.6 Algoritmi di apprendimento avanzati

L'algoritmo *stochastic gradient descent* costituisce la base di tutti gli algoritmi di apprendimento supervisionato su reti neurali. Esso però ha una problematica: con una grande variabilità di dati la possibilità di osservare un solo esempio conduce a modifiche dei pesi che non sempre conducono ad una rapida minimizzazione della funzione *loss*. Per visualizzare tale fenomeno, visualizziamo una curva di livello corrispondente ai valori di una funzione di costo L, variata rispetto a due pesi $\boldsymbol{W}_1 e \boldsymbol{W}_2$ della rete, valutata su tutto il training set.

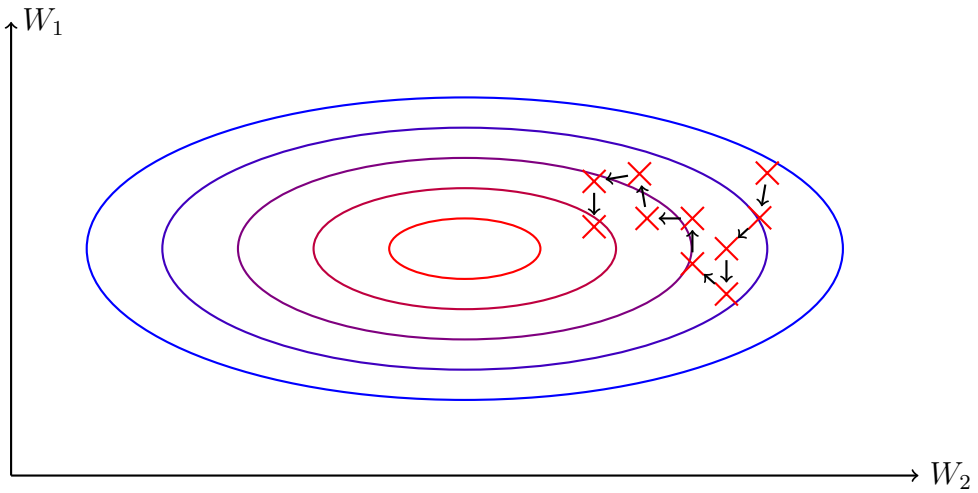


Figura 2.8: Esempio di applicazione di SGD

Dove le curve ellittiche rappresentano i curve di livello per valori decrescenti della funzione costo, con i valori evidenziati da un gradiente da rosso (valori bassi della funzione) al blu (valori alti della funzione). Sebbene la figura 2.8 sia stata costruita graficamente e non sia il frutto di una computazione reale, essa è utile per evidenziare il problema di *stochastic gradient descent*: sebbene esso converga ad un minimo globale, non è detto che ciascun passo di *backpropagation* avvicini la rete a tale minimo, in quanto viene preso in considerazione un solo esempio per volta. Si utilizza per questo un algoritmo detto *minibatch gradient descent*, che utilizza un numero prefissato

di esempi annotati (*minibatch*) per calcolare un valor medio del gradiente, applicandolo poi solo una volta che si sono esaminati tutti gli esempi.

Algoritmo 3 Minibatch Gradient Descent

Input: $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ parametrizzata da θ
Input: training set $I = (x_i, y_i), i \in [1, \ldots, d_s]$
Input: learning rate η
Input: loss function L
$i = 1$
while the halting conditions aren't met **do**
    $b = \{(x_i, y_i), \ldots, (x_{i+m}, y_{i+m})\}$
    $\hat{g} = 0$
    for all $i \in [i, i + m]$ **do**
        $\hat{g} \leftarrow \hat{g} + \nabla(L(f(x_i), y_i)$
        $i = (i + m) \mod d_{in}$
    end for
    $\theta \leftarrow \theta + \eta\hat{g}$
end while

Si noti come nell'esempio precedente si è supposto che il numero di *sample* d_s sia divisibile per da dimensione del *minibatch*. Se applichiamo tale tecnica alla figura di esempio precedente (che, ricordiamo, non contiene dati reali ma costituisce solo un possibile scenario dell'applicazione di sgd), utilizzando una dimensione del batch pari a 3, otteniamo:

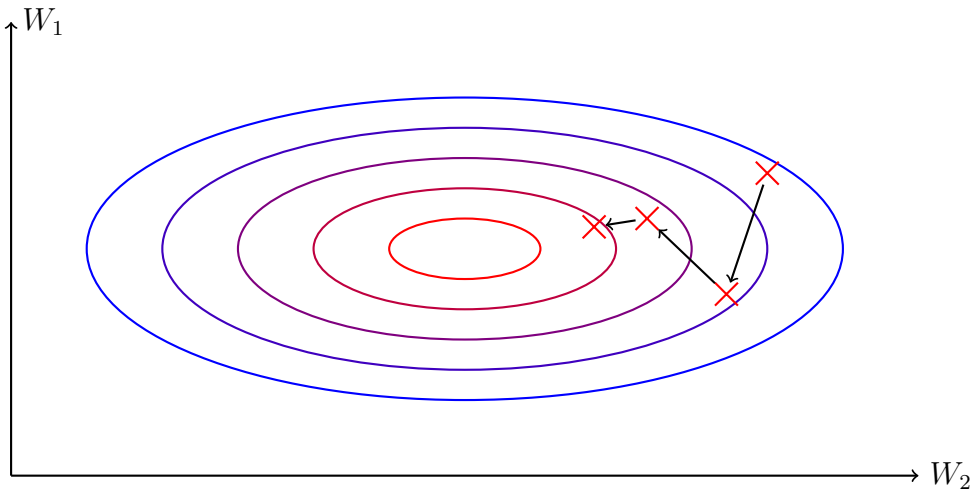


Figura 2.9: Esempio di applicazione di Minibatch Gradient Descent

Dove si è supposto di aver utilizzato nelle due figure uno *step* unitario per i gradienti, in modo da semplificare i calcoli. È evidente come un tale approccio riduca fortemente le "deviazioni" rispetto al minimo della funzione, risultando in progressi più rapidi.

È inoltre possibile, in modo analogo a ciò che si effettua in molti problemi di ottimizzazione, ridurre il *learning* rate man mano che l'algoritmo procede, in modo da effettuare via via spostamenti di magnitudo inferiore e quindi evitare situazioni in cui si oltrepassa il minimo globale.

Altre proposte di algoritmi di apprendimento utilizzano nozioni analoghe al momento in ambito fisico. In questo caso si tiene conto della direzione nella quale si sono spostati i pesi nelle iterazioni precedenti, in modo analogo a quanto si farebbe per applicare una forza ad un grave in movimento. Si riduce in questo modo la possibilità di oscillazioni rispetto al minimo globale della funzione costo. Fra tali proposte fra le più diffuse vi sono rmsprop [57] ed Adam [33].

2.1.7 Recurrent neural network e LSTM

Nell'ambito dell'elaborazione del linguaggio naturale è talvolta necessario utilizzare sequenze di lunghezza variabile, che non sono dunque utilizzabili da reti neurali tradizionali (*feed-forward*). Per gestire questo tipo di situazione esistono possibilità diverse: è possibilie infatti utilizzare reti neurali convoluzionali e *recurrent neural network*. Le reti neurali convoluzionali (*convolutional neural network*) sono in particolare molto utilizzate nell'ambito dell'elaborazione delle immagini e dei video. Nonostante sia possibile utilizzare tale strumento anche nell'ambito del linguaggio naturale, il solver da noi proposto non ne fa uso e pertanto non consideriamo utile una descrizione di tali reti in questa tesi.

Lo strumento che si è utilizzato per la gestione delle sequenze di lunghezza variabile è invece una rete basata su *Long Short Term Memory* (LSTM) [30]. Al fine di comprendere meglio le idee alla base di reti *recurrent* in generale, forniamo una definizione generale di livello ricorrente di una rete neurale. Formalmente, una *recurrent neural network* (RNN) è una rete che, data una sequenza di vettori di input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ ed un vettore rappresentante lo stato iniziale della rete $\boldsymbol{s}_0$, fornisce in output una lista di vettori stato $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$ e vettori di output $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$. È possibile esplicitare tale comportamento tramite due funzioni R ed O che operano rispettivamente sui vettori di stato e sui vettori di input:

$$RNN((\boldsymbol{s}_0.\boldsymbol{x}_{1:n}) = (\boldsymbol{s}_{1:n}, \boldsymbol{y}_{1:n}) \tag{2.35a}$$

$$\boldsymbol{s}_i = R(\boldsymbol{x}_{i-1}, \boldsymbol{x}_i) \tag{2.35b}$$

$$\boldsymbol{y}_i = O(s_i) \tag{2.35c}$$

Dove $\boldsymbol{x}_{1:n}, \boldsymbol{s}_{1:n}, \boldsymbol{y}_{1:n}$ indicano liste di vettori di indici compresi fra 1 ed n. A ciascun livello di una RNN sono poi associati dei pesi θ, che sono utilizzati per il calcolo delle coppie di outuput $(\boldsymbol{s}_i, \boldsymbol{y}_i)$.

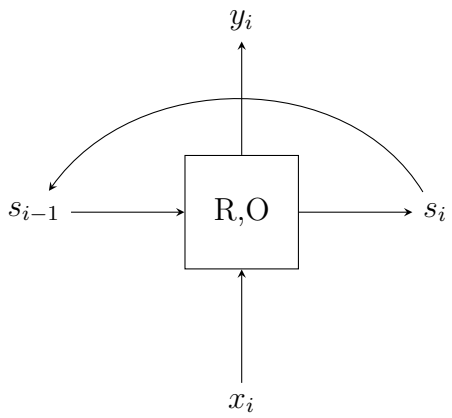


Figura 2.10: Rappresentazione ricorsiva di una RNN

Sebbene sia possibile interpretare una applicazione di una sequenza di input ad una RNN in modo ricorsivo (figura 2.10), durante la applicazione dell'algoritmo di *backpropagation* viene spesso svolta la rete nella sua applicazione ad un certo input (figura 2.11).

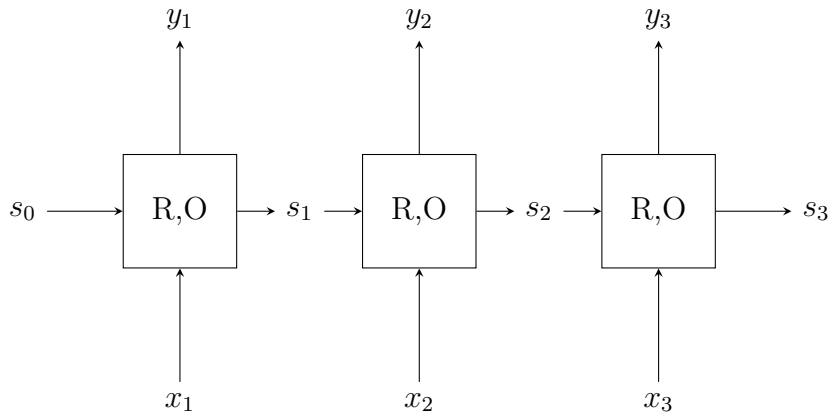


Figura 2.11: Rappresentazione ricorsiva di una RNN

In questo modo è possibile applicare tecniche di *backpropagation* alla rete come se essa fosse *feed-forward*. Questo tipo di tecnica è detto *backpropagation through time*.

Si è dunque discussa la struttura di base di livello ricorrente di una rete neurale. Non si è però fornita una formalizzazione di tali tipologie di rete. Descriveremo ora una tipologia di rete neurale detta *Long Short Term*

Memory(LSTM), che è in grado di mantenere informazioni su input che sono stati visti parecchie sequenze addietro, nonché informazioni più recenti. Formalmente, una LSTM è definita come:

$$LSTM(\boldsymbol{s}_{j-1}, \boldsymbol{x}_j) = (R_{LSTM}(\boldsymbol{s}_{j-1}, \boldsymbol{x}_j), O_{LSTM}(\boldsymbol{s}_j)) \tag{2.36a}$$

$$\boldsymbol{s}_j = R_{LSTM}(\boldsymbol{s}_{j-1}, \boldsymbol{x}_j) = (\boldsymbol{c}_j, \boldsymbol{h}_j) \tag{2.36b}$$

$$\boldsymbol{c}_j = \boldsymbol{c}_{j-1} \odot \boldsymbol{f} + \boldsymbol{g} \odot \boldsymbol{i} \tag{2.36c}$$

$$\boldsymbol{h}_j = \tanh(c_j) \odot \boldsymbol{o} \tag{2.36d}$$

$$\boldsymbol{i} = \sigma(\boldsymbol{x}_j \boldsymbol{W}^{xi} + \boldsymbol{h}_{j-1} \boldsymbol{W}^{hi}) \tag{2.36e}$$

$$\boldsymbol{f} = \sigma(\boldsymbol{x}_j \boldsymbol{W}^{xf} + \boldsymbol{h}_{j-1} \boldsymbol{W}^{hf}) \tag{2.36f}$$

$$\boldsymbol{o} = \sigma(\boldsymbol{x}_j \boldsymbol{W}^{xo} + \boldsymbol{h}_{j-1} \boldsymbol{W}^{ho}) \tag{2.36g}$$

$$\boldsymbol{g} = \tanh(\boldsymbol{x}_j \boldsymbol{W}^{xg} + \boldsymbol{h}_{j-1} \boldsymbol{W}^{hg}) \tag{2.36h}$$

$$O_{LSTM}(\boldsymbol{s}_j) = \boldsymbol{h}_j \tag{2.36i}$$

$$\boldsymbol{s}_j \in \mathbb{R}^{2d_h}; \boldsymbol{x}_i \in \mathbb{R}^{d_x}; \boldsymbol{c}_j, \boldsymbol{h}_j, \boldsymbol{i}, \boldsymbol{f}, \boldsymbol{o}, \boldsymbol{g} \in \mathbb{R}^{d_h}; \boldsymbol{W}^{xo} \in \mathbb{R}^{d_x \times d_h}; \boldsymbol{W}^{ho} \in \mathbb{R}^{d_h \times d_h} \tag{2.36j}$$

Dove il simbolo $\odot$ viene utilizzato per indicare il prodotto di Hadamard. Lo stato j-esimo è composto dai vettori $\boldsymbol{c}_i$ e $\boldsymbol{h}_j$ (2.36b), detti rispettivamente *memory component* e *output/state component*. Vengono inoltre utilizzati tre *gate*, che controllano quali informazioni debbano essere mantenute per ciascuna sequenza di input. Essi sono l'*input gate* $\boldsymbol{i}$ (2.36e), il *forget gate* $\boldsymbol{f}$ (2.36f) e l'*output gate* $\boldsymbol{o}$ (2.36g). I valori dei singoli *gate* vengono calcolati sulla base di combinazioni lineari dell'input corrente $\boldsymbol{x}_k$ e di $\boldsymbol{h}_{j-1}$, a cui viene poi applicata una funzione di attivazione sigmoide. Un candidato per l'aggiornamento dello stato della rete $\boldsymbol{g}$ viene poi calcolato mediante una combinazione lineare di $\boldsymbol{x}_i$ e $\boldsymbol{h}_{j-1}$, a cui viene applicata una tangente iperbolica come funzione di applicazione (2.36h). Si aggiorna dunque la memoria $\boldsymbol{c}_j$: il *forget gate* controlla quali informazioni vengano mantenute dalla memoria precedente della rete, mentre l'*input gate* controlla quante informazioni dell'input

corrente vengano mantenute (2.36c). Infine si determina il valore di $\boldsymbol{h}_j$ viene calcolato a partire dal contenuto della memoria $\boldsymbol{c}_j$, passato attraverso una tangente iperbolica e controllato dall'*output gate* $\boldsymbol{o}$ (2.36d).

È evidente come l'utilizzo di *gate* separati per input, output e memoria interna della rete consenta alle LSTM di mantener in memoria informazioni rilevanti che sono state viste parecchie sequenze di input addietro e non risentire del problema del "*vanishing gradient*", mentre vengono scartate informazioni recenti ma poco significative. È grazie a questa struttura che LSTM ha ottenuto grandi successi in ambiti diversi e risulta essere il modello più comune per la creazione di *recurrent neural network*.

2.1.8 Word embedding: Word2vec

Si sono finora discusse le componenti principali utilizzate dalla rete neurale sfruttata dal nostro *coreference solver*. Vi è però uno strumento fondamentale, anch'esso basato su reti neurali, del quale non si è fornita una descrizione. È infatti lecito chiedersi come si possano rappresentare in forma vettoriale le parole espresse in un linguaggio naturale, in modo da consentirne l'utilizzo da parte di reti neurali. Uno degli approcci più utilizzati negli ultimi anni è basato sugli algoritmi word2vec sviluppati per conto di Google da un team guidato da Mikolov [43].

Word2vec si pone l'obiettivo di rappresentare le parole in uno spazio vettoriale che mantenga informazioni di tipo semantico e sintattico. L'approccio utilizzato si basa su apprendimento non supervisionato di reti neurali a cui viene fornito un testo nella lingua per cui si desiderano ottenere gli *embedding*, utilizzando due modelli distinti, detti CBOW e skip-gram.

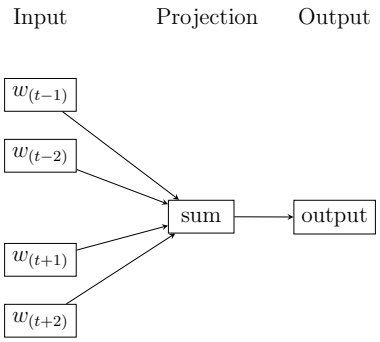


Figura 2.12: CBOW

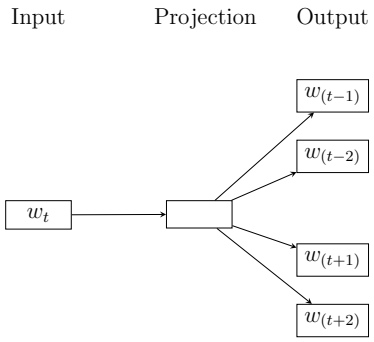


Figura 2.13: Skip-gram

L'approccio CBOW (figura 2.12) si basa sull'utilizzo delle parole circostanti un termine (dette contesto) per prevedere la parola in questione. Ciascuna parola viene codificata come un vettore *one-hot* di dimensione pari al vocabolario individuato nel testo fornito in input. Tali valori vengono poi utilizzati da un livello nascosto della rete neurale per apprendere le rappresentazioni vettoriali delle parole. Tali *embedding* vengono poi sommati in un livello intermedio della rete, che viene infine collegato tramite un livello *feed-forward* con *bias* e funzione di attivazione softmax ad un vettore di output che è un *encoding one-hot* analogo a quello fornito in input. La rete viene poi allenata sul testo, in modo da apprendere in modo autonomo le rappresentazioni vettoriali delle parole.

L'approccio skip-gram (figura 2.13) funziona invece in modo inverso rispetto a CBOW. Viene fornita una parola in input, rappresentata da un *encoding one-hot* analogo a quanto visto in precedenza. Tale *encoding* attraversa poi un livello nascosto di dimensione pari all'embedding desiderato. Infine si utilizzano un numero di output pari alla dimensione del vocabolario per ottenere le probabilità che in una posizione di un contesto di dimensione fissata sia contenuta una certa parola del vocabolario. Vengono selezionate come parole del contesto su cui effettuare la misura della loss della rete un sottoinsieme delle parole del contesto scelte in nodo casuale. L'*embedding* viene dunque estratto dal livello nascosto della rete.

Tali approcci hanno una serie di proprietà interessanti. Se si estraggo-

no infatti i termini più vicini ad una certa parola nello spazio vettoriale in termini di coseno di similitudine essi risultano solitamente essere semanticamente prossimi alla parola stessa. È inoltre possibile effettuare operazioni di addizione e sottrazione per i vettori con risultati intuitivamente attesi dal punto di vista semantico:

$$E(\text{Roma}) - E(\text{Italia}) + E(\text{Francia}) \approx E(\text{Parigi}) \tag{2.37a}$$

$$E(\text{Zio}) - E(\text{Maschio}) + E(\text{Femmina}) \approx E(\text{Zia}) \tag{2.37b}$$

Dove la relazione $\approx$ indica in questo caso il vettore di *embedding* più prossimo rispetto alla parola in termini di coseno di similitudine, mentre E indica l'*embedding* associato ad un certo termine. Dal punto di vista pratico, CBOW risulta essere più veloce ma skip-gram fornisce *embedding* di miglior qualità per termini con meno occorrenze nel testo. Si è pertanto scelto quest'ultimo modello per l'*embedding* dei termini utilizzati dal nostro solver.

Si sono introdotte in questo capitolo le principali componenti delle reti neurali e si è discusso di *recurrent neural network* e di word2vec. Si procede ora ad una descrizione di alcuni algoritmi per la *coreference resolution* implementati tramite reti neurali *deep*, la cui descrizione non è stata presentata in precedenza per consentire una miglior comprensione del loro funzionamento grazie ai prerequisiti descritti in questo capitolo.

2.2 Coreference resolution tramite Reti Neurali Deep

Oltre agli approcci già visti in precedenza, negli ultimi anni una delle tecnologie più interessanti per quanto riguarda l'elaborazione del linguaggio naturale in generale e nello specifico la *coreference resolution* è l'utilizzo di reti neurali *deep*. Se la trattazione completa delle tecnologie alla base del cosiddetto *deep learning* sarà oggetto di un capitolo successivo, è comunque interessante una trattazione di alcune soluzioni molto recenti che risultano avere prestazioni

pari allo stato dell'arte per il task in esame. Le reti neurali sono infatti lo strumento fondamentale che è stato utilizzato durante lo sviluppo del nostro solver, tenendo indubbiamente conto delle proposte presenti in letteratura.

2.2.1 Algoritmo di Clark, Manning

La proposta di Clark,Manning [14] è a nostra conoscenza la prima implementazione di un solver per la *coreference* basato su reti neurali *deep*. Uno degli principali obiettivi di tale proposta è quello di dimostrare come non sia necessario l'utilizzo di *feature* numerose e complesse per costruire un sistema con prestazioni allo stato dell'arte. Tale sistema utilizza tre componenti:

- **Mention-Pair Encoder**: si tratta di uno strumento che fornisce una rappresentazione per ciascuna coppia di menzioni, passando informazioni rilevanti ad una rete *feed-forward*;
- **Cluster-Pair Encoder**: produce rappresentazioni di cluster di menzioni effettuando un'operazione di *pooling* su coppie di menzioni rispettivamente appartenenti ai due cluster;
- **Mention-Ranking Model**: fornisce un punteggio per ciascuna coppia di menzioni utilizzando queste ultime come input di una rete neurale. I parametri così ottenuti vengono utilizzati per inizializzare il *cluster-pair encoding* e per effettuare pruning su quali cluster considerare per un *merge*.

Il sistema fornisce al *mention-pair encoder* una serie di *feature*, nelle quali si utilizza la media fra *word embedding* per fornire la rappresentazione di una sequenza di parole, oltre ad vettori *one-hot binned* che codificano le distanze fra parole. Le feature utilizzate sono:

- **Embedding Features**: i *word embedding* della prima e dell'ultima parola della menzione, del nodo padre in un albero di *parsing* a dipendenze, della *head word*, due parole precedenti e due parole successive;

- **Additional Mention Features**: Il tipo della menzione (pronome, nominale, nome proprio o lista), la posizione della menzione (indice della menzione diviso il numero di menzioni nel documento), se la menzione è contenuta in un'altra menzione, il numero di parole della menzione;

- **Document genre**: il genere del documento considerato;

- **Distance Features**: la distanza fra le due menzioni in termini di frasi;

- **Speaker Features**: se le due menzioni hanno lo stesso speaker e se una delle due menzioni è lo speaker dell'altra frase secondo le regole di *string matching* proposte da Raghunathan et al [51];

- **String Matching Features**: *head match*, *string match* esatto e *string match* parziale.

Tali *feature* vengono poi concatenati in un vettore $\boldsymbol{h}_0$, che costituisce l'input per il *mention pair encoder*. Tale rete utilizza poi tre *hidden layer* di unità ReLU completamente connesse al livello precedente:

$$\boldsymbol{h}_i(a, m) = max(0, \boldsymbol{W}_i \boldsymbol{h}_{i-1}(a, m) + \boldsymbol{b}_i) \tag{2.38}$$

Dove $\boldsymbol{W}_i$ è la matrice dei pesi del livello i-esimo della rete, mentre $\boldsymbol{b}_i$ è il vettore di bias. L'output dell'ultimo livello nascosto che rappresenta la coppia di menzioni (a, m): $\boldsymbol{r}_m(a, m) = \boldsymbol{h}_3(a, m)$.

La seconda componente del sistema, il *cluster-pair encoder*, effettua operazioni di *pooling* su due cluster di menzioni $c_i = m_1^i, m_2^i, \ldots, m_{|c_i|}^i$ e $c_j = m_1^j, m_2^j, \ldots, m_{|c_j|}^j$. Si utilizza dunque l'insieme di rappresentazioni di tutte le coppie di menzioni dei due cluster ottenute tramite *mention-pair encoder*:

$$\boldsymbol{R}_m(c_i, c_j) = [\boldsymbol{r}_m(m_1^i, m_1^j), \boldsymbol{r}_m(m_1^i, m_2^j), \ldots, \boldsymbol{r}_m(m_{|c_i|}^i, m_{|c_j|}^j)] \tag{2.39}$$

Per ottenere la rappresentazione dei due cluster $\boldsymbol{r}_c(c_i, c_j)$ A tali coppie di menzioni viene infatti applicato sia un *max pooling*, sia un *average pooling*,

concatenando poi i due risultati:

$$\boldsymbol{r}_c(c_i, c_j)_k = \begin{cases} max(\boldsymbol{R}_m(c_i, c_j)_k) & \text{per } 0 \leq k < d \\ avg(\boldsymbol{R}_m(c_i, c_j)_{k-d}) & \text{per } d \leq k < 2d \end{cases} \tag{2.40}$$

Viene infine utilizzato un classificatore lineare sulle rappresentazioni del *mention-pair encoder* per effettuare il *mention-ranking*:

$$s_m(a, m) = \boldsymbol{W}_m \boldsymbol{r}_m(a, m) + \boldsymbol{b}_m \tag{2.41}$$

Dove $\boldsymbol{W}_m$ è la matrice dei pesi della rete e $\boldsymbol{b}_m$ è il vettore di bias.

Il *mention-ranking model* utilizza una funzione di *loss* proposta da Wiseman et al [64] e detta *slack rescaled max margin training objective*. Supponendo che il training set sia composto dalle menzioni $m_1, m_2, \ldots, m_n$, si denota con $A(m_i)$ l'insieme di candidati ad essere antecedenti della menzione m_i. Con t_m si indica l'insieme dei candidati che sono antecedenti di m_i, oppure $\{NA\}$ se m_i non ha antecedenti. L'antecedente con il miglior punteggio stimato dalla rete viene indicato con $\hat{t}_i$ ed è definito come:

$$\hat{t}_i = \underset{t \in T(m_i)}{\arg\max}\, s_m(t, m_i) \tag{2.42}$$

La loss è dunque data da:

$$L = \sum_{i=1}^{N} \max_{a \in A(m_i)} \Delta(a, m_i)(1 + s_m(a, m_i) - s_m(\hat{t}, m_i)) \tag{2.43}$$

Dove Δ è una funzione che determina il tipo di errore effettuato, consentendo

una taratura dei pesi più granulare:

$$\Delta(a, m_i) = \begin{cases} \alpha_{FN} & \text{se } a = NA \wedge T(m_i) \neq NA \\ \alpha_{FA} & \text{se } a \neq NA \wedge T(m_i) = NA \\ \alpha_{WL} & \text{se } a \neq NA \wedge a \notin T(m_i) \\ 0 & \text{se } a \in T(m_i) \end{cases} \tag{2.44}$$

dove i tre pesi non nulli corrispondono rispettivamente ai casi di *false new*, *false anaphoric*, *wrong link*.

Oltre ai modelli precedenti, che operano su coppie di menzioni, è utile utilizzare un cluster-ranking model s_c, che fornisca uno score a due cluster in modo da comprendere quali siano valutati più compatibili dalla rete. Esso è strutturato in maniera del tutto analoga a quanto visto per il *mention-ranking model*.

$$s_c(c_i, c_j) = \boldsymbol{W}_c \boldsymbol{r}_c(c_i, c_j) + \boldsymbol{b}_c \tag{2.45}$$

Viene infine utilizzata un'ulteriore rete per fornire una misura della probabilità che una menzione abbia un antecedente, s_{NA}, definita nel modo seguente:

$$s_{NA}(m) = \boldsymbol{W}_{NA} \boldsymbol{r}_m(NA, m) + b_{NA} \tag{2.46}$$

Utilizzando il *ranking* dei *cluster* ed il *ranking* delle menzioni con antecedenti, l'algoritmo seleziona per ciascuna menzione l'azione da compiere (non unirla ad alcun cluster oppure cercare un cluster a cui unirla fra i candidati). Per selezionare il cluster fra i candidati viene utilizzato un algoritmo *easy-first*, nel quale si utilizza il *mention-ranking model* per valutare sempre le scelte più semplici per prime. Vengono inoltre ignorati i candidati con score troppo bassi, riducendo la complessità dell'algoritmo.

Nella fase di training viene infine utilizzato un algoritmo del tipo *learning-to-search*, in quanto lo stato del sistema viene variato con ciascuna iterazione dell'algoritmo.

Si è dunque descritta una prima proposta di algoritmo basato su reti

neurali *deep*, che è in grado di ottenere risultati stato dell'arte utilizzando poche e semplici *feature*. Tramite questo algoritmo vengono dimostrate le potenzialità delle reti neurali *deep* per il task della *coreference*.

2.2.2 Una proposta basata su Recurrent Neural Network

Dopo aver discusso la proposta di Clark e Manning è interessante esaminare un approccio alternativo basato su *Recurrent Neural Network* (RNN). Tale tipologia di rete, descritta nella sezione 2.1.7, ha il grosso vantaggio di poter essere applicata a sequenze di lunghezza arbitraria. Si tratta infatti di reti che mantengono una memoria degli input precedenti, fornendo infine una rappresentazione della sequenza di input con dimensioni prefissate. Il vantaggio principale di un tale approccio, rispetto ad operazioni di *pooling* per controllare la dimensionalità delle sequenze in input, è il fatto che vengono mantenute la rete tende a mantenere le informazioni riguardanti l'ordine in cui sono stati forniti gli input. La proposta di Wiseman et al. [63], in particolare, fa utilizzo di LSTM per fornire una rappresentazione di un cluster di menzioni, utilizzabile poi da un *coreference resolutor*.

Si procede prima di tutto rappresentando ciascuna menzione tramite un insieme di indicatori in un vettore binario, in modo analogo a quanto visto per l'algoritmo di Clark e Manning. Viene dunque usata una funzione di *embedding* non lineare, che mappa ciascuna menzione ad una sua rappresentazione in uno spazio vettoriale $\boldsymbol{h}_c(x)$:

$$\boldsymbol{h}_c(x) = \tanh(\boldsymbol{W}_c\phi_a(x) + \boldsymbol{b}_c) \tag{2.47}$$

Dove $\boldsymbol{W}_c$ e $\boldsymbol{b}_c$ sono rispettivamente i pesi e il *bias* della rete.

Si procede dunque fornendo in sequenza le menzioni di un cluster alla RNN, ottenendo via via degli stati nascosti della rete della forma:

$$\boldsymbol{h}_j^{(m)} = RNN(\boldsymbol{h}_c(X_j^{(m)}), \boldsymbol{h}_{j-1}^{(m)}) \tag{2.48}$$

Dove $\boldsymbol{h}_j^{(m)}$ è la j-esima menzione del cluster m.

Alla base del modello vi sono due componenti: un termine locale, che definisce una compatibilità *pair-wise* fra coppie di menzioni, ed un termine globale, che utilizza funzioni basate sullo stato corrente dei cluster. Il candidato che verrà selezionato è dunque:

$$\arg\max_{y_1,\ldots,y_N} \sum_{n=1}^{N} f(x_n, y_n) + g(x_n, y_n, z_{1:n-1}) \tag{2.49}$$

In questa equazione la componente locale è rappresentata dalla funzione f, mentre g rappresenta la componente globale. In particolare, f è implementata da una rete neurale *feed-forward* già proposta dagli autori in [64], che utilizza una casistica particolare nel caso in cui si operi sul *placeholder* ϵ, che testa la non anaforicità della menzione in esame:

$$f(x,y) = \begin{cases} \boldsymbol{u}^T \begin{bmatrix} \boldsymbol{h}_a(x) \\ \boldsymbol{h}_p(x,y) \end{bmatrix} + u_0 & \text{se } y \neq \epsilon \\ \boldsymbol{v}_t \boldsymbol{h}_a(x) + v_0 & \text{se } y = \epsilon \end{cases} \tag{2.50}$$

Dove u, v e b_0, b_1 sono i parametri del modello, h_a ed h_p sono *embedding* rispettivamente del contesto locale della menzione e dell'affinità fra le due menzioni. Tali *embedding* sono costruiti in modo simile ad h_c ed espressi come:

$$\boldsymbol{h}_a(x) = tanh((\boldsymbol{W})_a \phi_a(x) + \boldsymbol{b}_a) \tag{2.51a}$$

$$\boldsymbol{h}_p(x) = tanh((\boldsymbol{W})_p \phi_p(x,y) + \boldsymbol{b}_p) \tag{2.51b}$$

Dove Φ_a e Phi_p sono rispettivamente *feature* riguardanti il contesto della menzione x e l'affinità delle due menzioni x ed y. Si indica con $\boldsymbol{h}_{<n}^{(m)}$ lo stato nascosto della RNN dopo aver consumato tutte le menzioni del cluster prima di x_n. Viene dunque specificata una funzione g che fornisca uno score globale

basato su tutte le decisioni effettuate nel tempo:

$$g(x_n, y_n, z_{1:n-1}) = \begin{cases} \boldsymbol{h}_c(x_n)^T \boldsymbol{h}_{<n}^{(z_{y_n})} & \text{se } y \neq \epsilon \\ NA(x_n) & \text{se } y \neq \epsilon \end{cases} \tag{2.52}$$

Dove la funzione NA fornisce un punteggio per l'assegnamento di ϵ tramite una funzione non lineare di tutti gli stati nascosti dei cluster:

$$NA(x_n) = \boldsymbol{q}^T \tanh(\boldsymbol{W}_s \begin{bmatrix} \phi_a(x) \\ \sum_{m=1}^{M} h_{<n}^m \end{bmatrix} + \boldsymbol{b}_s) \tag{2.53}$$

Tale meccanismo di valutazione globale si basa sull'intuizione che, al fine di valutare se y_n sia un buon antecedente per x_n, si debba aggiungere al punteggio finale un termine globale, che valuti la coerenza di x_n con le menzioni già parte del cluster a cui appartiene y_n, ovvero $X^{(z_{y_n})}$. Questo punteggio viene espresso tramite un *dot product* fra l'*embedding* della menzione ed il vettore rappresentante lo stato della RNN prima della decisione su tale menzione. La funzione NA è invece utilizzata quando si intenda stabilire se la menzione corrente sia non anaforica. A tal fine è utile esaminare lo stato di tutti i cluster già costruiti fino ad ora, poiché essi forniscono informazioni su altri insiemi singoletto (termini non anaforici) ed informazioni su altri possibili antecedenti.

Il training avviene dunque in modo end-to-end sul problema della *coreference*, utilizzando la *backpropagation*. A tal fine è possibile precalcolare i cluster forniti da un oracolo, ovvero i cluster che sono stati costruiti prima dell'assegnamento della menzione corrente x_n. Se è possibile precalcolare i cluster tramite l'oracolo, non è dato sapere quale menzione particolare verrà selezionata come antecedente di x_n: all'interno dello stesso cluster di coreferenti corretto potrebbe venire selezionata una qualsiasi menzione. Per questa ragione nella funzione obiettivo viene utilizzato un "*latent antecedent*", ovvero l'antecedente migliore nel cluster corretto per ciò che viene valutato dalla rete.

L'algoritmo di *clustering* utilizzato è di tipo *greedy*, in quanto se l'obiettivo locale ha complessità quadratica, la risoluzione del problema di *clustering* globale è NP-hard. Il *clustering* avviene dunque in modo analogo ad un modello *mention-ranking* visto in 1.6.3, che tuttavia aggiorna ad ogni nuovo assegnamento di menzioni lo stato globale dei cluster ed utilizza anche il termine di compatibilità globale g nella sua valutazione degli antecedenti.

L'approccio proposto, sebbene ottenga risultati leggermente inferiori a quelli di Clark e Manning sul corpus del task CoNLL 2012, costituisce la prima applicazione di RNN al problema della *coreference* che abbia avuto successo. Si è dimostrato infatti come l'utilizzo di RNN sia in grado di rappresentare lo stato dei cluster costruiti dall'algoritmo a ciascun passo, sfruttando quindi informazioni globali, che tengano conto dell'ordine delle menzioni nel documento, a differenza di quanto visto nell'approccio di Clark e Manning, che utilizzava operazioni di *pooling* che non mantengono questo tipo di informazioni.

Capitolo 3

Coref Resolver: un solutore basato su reti neurali deep

Nei capitoli precedenti si è dapprima fornita una descrizione del fenomeno linguistico della *coreference* e di alcuni strumenti per la sua risoluzione. Si è poi discusso di reti neurali e delle loro principali componenti, nonché di *recurrent neural network* e word2vec, strumenti comunemente utilizzati nell'ambito dell'elaborazione del linguaggio naturale tramite reti neurali *deep*. Si sono infine presentati alcuni *solver* per la *coreference* basati su tale tecnologia.

Si procede dunque alla descrizione del solver proposto in questa tesi e degli strumenti utilizzati per la sua implementazione. Si vedrà come tale *solver*, che non è in grado, allo stato attuale, di ottenere performance eccessivamente competitive con lo stato dell'arte, costituisca comunque un esperimento interessante, che sfrutta reti neurali ricorrenti per codificare le singole menzioni in una *coreference chain* in maniera inedita.

Durante lo sviluppo si sono implementate e testate varie iterazioni del *solver*, che hanno ottenuto man mano risultati migliori delle versioni precedenti, di cui si darà conto solo in parte all'interno di questo capitolo, che si concentrerà invece sull'ultima versione sviluppata.

3.1 Ontonotes e formato CoNLL

Si è discusso nella sezione 1.7 dei vari corpora per il task della *coreference* che sono stati compilati nel tempo. Recentemente il *dataset* più comune per l'implementazione e la comparazione di *solver* per tale task è un sottoinsieme di OntoNotes utilizzato nel task di *coreference* CoNLL del 2012. Nell'implementazione del *solver* si è ottenuto l'accesso al *dataset* OntoNotes nella sua interezza, che viene però fornito nella forma di file differenti per ciascuna informazione linguistica che essi contengono. La *coreference* viene ad esempio codificata in file xml con *document type* definition ad-hoc, così come le *named entities*. Il *parse tree* è invece memorizzato in file che utilizzano un formato differente. Per semplificare il *parsing* dei documenti di OntoNotes, durante le campagne di valutazione CoNLL è stato fornito un formato che contiene tutte le informazioni rilevanti per ciascun documento in un singolo file. In particolare, vengono fornite informazioni annotate manualmente su *Part of Speech* della parola, *parse tree* sintattico, *named entities* e *coreference chain*.

Si è dunque scelto di utilizzare tali file, che sono stati convertiti dal corpus OntoNotes tramite uno script ottenuto in [1]. Tale script fornisce inoltre una segmentazione del corpus in *training set*, *validation set*, *test set* e, *conll-2012-test*, un *test set* che aderisce perfettamente a quello utilizzato nella campagna di valutazione CoNLL 2012. Tali file sono inoltre stati forniti al modulo utilizzato per il *preprocessing*, che utilizza il *parser* per il formato CoNLL fornito da stanford CoreNLP.

Il *solver* sfrutta infine file in formato hdf5 [26], manipolati tramite la libreria h5py [15], nei quali vengono memorizzate informazioni sulle menzioni, estratte tramite *mention extractor*, nonché le altre informazioni sui documenti, ottenute dal *parsing* dei file in formato CoNLL.

Il formato CoNLL è stato inoltre utilizzato per salvare su disco le *coreference* individuate in fase di *testing*, in modo da consentirne la valutazione tramite lo script *reference coreference scorers* per il calcolo di varie metriche per la a partire dal formato CoNLL [41, 50].

3.2 Strumenti utilizzati

Prima di fornire una descrizione precisa del *solver* utilizzato e della struttura della rete neurale di cui esso fa uso, è necessaria una breve menzione degli strumenti utilizzati da parte del *solver*. Esso è stato implementato nel linguaggio di programmazione python, che fornisce librerie per la manipolazione dei dati e per la creazione di reti neurali di semplice utilizzo ma che offrono grandi potenzialità per l'utilizzo in programmi che manipolino dati complessi in modo intuitivo e sintetico.

Oltre a tali librerie ed al *solver* python, si è utilizzato un piccolo modulo Java che sfrutta la libreria CoreNLP di Stanford [42] per estrarre i candidati a divenire menzioni ed alcune loro *feature*, tramite una pipeline di analisi del linguaggio naturale completa fornita da tale libreria.

3.2.1 Keras e TensorFlow

La libreria utilizzata per l'implementazione della rete neurale è keras [13], una libreria di alto livello per l'implementazione di reti neurali. Tale libreria fornisce una serie di livelli di reti neurali predefiniti, con la possibilità di configurare vari aspetti del loro funzionamento (funzione di attivazione, dimensione, etc) senza però formalizzare le operazioni matriciali della rete in modo esplicito. La possibilità di comporre in modo semplice ed immediato tali livelli consente iterazioni rapide ed efficaci di diverse reti, risultando essere uno strumento molto utile per effettuare sperimentazioni nell'ambito del *deep learning*. Oltre a diverse tipologie di livelli di reti neurali disponibili, che comprendono reti convoluzionali e *recurrent*, la libreria fornisce ottimizzatori e funzioni di *loss* avanzate, che possono essere sostituite modificando un solo parametro della rete.

Per la costruzione delle reti Keras sfrutta librerie di più basso livello in modo totalmente trasparente allo sviluppatore. È possibile infatti utilizzare TensorFlow, CNTK o Theano. Fra queste librerie si è scelto durante l'implementazione della rete di utilizzare TensorFlow [2], una libreria sviluppata da

Google che è però *open source*. Si è scelta tale libreria in quanto la comunità ad essa associata pare essere molto attiva e la libreria, in costante evoluzione, fornisce strumenti molto utili, quali TensorBoard, che consente di visualizzare molteplici aspetti associati alla rete, dalla sua struttura all'andamento della sua accuratezza durante il *training*.

3.2.2 Corenlp

Al fine di consentire l'estrazione delle menzioni, si è implementato un modulo Java, che fa uso della libreria CoreNLP di Stanford. Tale libreria fornisce una pipeline che implementa i principali strumenti di elaborazione del linguaggio naturale, compresi vari *coreference solver*. Si è dunque utilizzato il mention extractor del *solver* di CoreNLP più recente (di cui si è discusso in 2.2.1), che fornisce la possibilità di estrarre menzioni direttamente dai file in formato CoNLL. Si sono poi estratte menzioni con tre modalità diverse:

- **Golden mention**: le menzioni appartenenti a *coreference chain* annotate nel corpus;
- **Golden parse mention**: menzioni estratte in modo automatico a partire dall'albero sintattico e dalle *named entities* annotate nel corpus;
- **Auto mention**: menzioni estratte in modo automatico utilizzando il *parser* e il riconoscitore di *named entity* di corenlp.

Si sono inoltre memorizzate informazioni di varia natura sulle menzioni:

- **Genere**: maschile, femminile, neutro o ignoto;
- **Numero**: singolare, plurale o ignoto;
- **Animacy**: animato, inanimato o ignoto;
- **Indice head**: l'indice (nella frase) della *head* della menzione;
- **Tipo di menzione**: lista, pronominale, proper o nominale;

- **Tipo di named entity**: varie tipologie a seconda della metodologia utilizzata per estrarre le menzioni (*golden mention*, *golden parse*, *auto mention*);

- **Persona**: *I*, *you*, *he*, *she*, *it*, *we*, *they*, *unknown*.

Tali informazioni sono poi state utilizzate come *feature* della rete neurale. Rispetto alla tipologia delle *named entity*, si sono selezionate classi ottenute in tutte le modalità di estrazione delle menzioni, ovvero "O" (che indica il fatto che la menzione non corrisponda ad una *named entity*), persone, luoghi, organizzazioni ed altro.

L'output del sistema viene poi serializzato in un file json, che sarà successivamente letto dal *solver*, in modo da memorizzare le menzioni nei file hdf5 che esso utilizza durante l'esecuzione.

3.3 Mention-pair model modificato

Si è discusso in precedenza (sezione 1.6) dei vari paradigmi possibili per l'implementazione di un sistema di coreference basato su *machine learning*. Rispetto a tali classi, il *solver* proposto utilizza una versione modificata del modello mention-pair, che opera su coppie di menzioni, alla ricerca dell'antecedente di ciascuna menzione.

In particolare, si è scelto di fornire alla rete informazioni sull'entità (il cluster di menzioni) a cui appartiene il candidato antecedente corrente, ovvero una lista di *word embedding* delle altre menzioni dell'entità, sulla cui natura discuteremo in seguito. Tali embedding sono stati ottenuti tramite l'algoritmo *skip-gram* di word2vec, che è stato lanciato su un *dump* di wikipedia inglese, a cui è stato concatenato un *dump* del training set. Le parole sconosciute presenti in *validation* e *test set* sono poi state rappresentate tramite un embedding random. Infine, poiché la presenza di vettori con norma due troppo elevata crea problemi di condizionamento alle reti neurali, tali vettori sono stati normalizzati per avere norma unitaria.

Il *clustering* avviene poi secondo una variante dell'algoritmo *best-first*, che seleziona il candidato con la maggior probabilità stimata dal sistema di essere l'antecedente della menzione corrente. L'algoritmo da noi utilizzato consente di ridurre il numero di candidati antecedenti considerati nel caso in cui esso sia vicino alla menzione corrente, considerando inoltre la distanza dalla menzione come fattore importante per la selezione degli antecedenti. Procediamo dunque alla descrizione di tale approccio, a partire dalla creazione di una lista di liste di candidati, che costituisce una lista di batch di candidati, popolati a ritroso nel documento. Nello pseudocodice seguente si utilizzano le primitive append(L,e), che aggiungono l'elemento e alla lista L, e $|L|$, che fornisce la lunghezza della lista L. Le liste vuote vengono inizializzate come $[]$.

Algoritmo 4 Costruzione della lista di batch di candidati

Input: Current mention m_c
Input: Current document mentions d_m
$i_p \leftarrow$ the index of the phrase containing m_c
$L \leftarrow []$
$B \leftarrow []$
for $i \leftarrow i_p; i >= 0; i \leftarrow i - 1$ **do**
    for all $m \in d_m \,|\, m$ preceeds $c_m \wedge m \in$ the i-th phrase **do**
        append(B,m)
    end for
    if $|B| \geq 10$ **then**
        append(L,B)
        $B \leftarrow []$
    end if
end for
if $|B| \neq 0$ **then**
    if $|L| \neq 0$ **then**
        append the elements of B to the last list contained in L
    else

19: append(L,B)
20: **end if**
21: **end if**
22: **return** L

Tale algoritmo fornisce dunque una lista di liste di candidati, che sono ordinati per distanza (in termini di frasi) crescente dalla menzione corrente. Si noti inoltre come le menzioni di una stessa frase siano sempre parte di una stessa lista, in modo da imporre al sistema di considerare sempre tutte le menzioni di una stessa frase.

L'algoritmo procede dunque valutando ciascun batch nella lista di liste di candidati, fornendoli come input alla rete neurale e valutando la probabilità che ciascuno di essi sia l'antecedente della menzione corrente. Nello pseudocodice seguente si fa uso, oltre alle primitive discusse in precedenza, della primitiva popfirst(L), che rimuove e ritorna il primo elemento della lista L, e della notazione $L[i]$, che indica l'i-esimo elemento della lista L.

Algoritmo 5 Trova antecedente

1: **Input**: current mention m_c
2: **Input**: current mention candidates list of batches L
3: **while** $|L| > 0$ **do**
4: $B \leftarrow$popfirst(L)
5: $R \leftarrow$ a list of estimated probabilities that each of the candidates in B is an antecedent of m_c
6: $i_m \leftarrow \arg\max_i(R[i])$
7: **if** $R[i_m] \geq 0.5$ **then**
8: **return** $B[i]$
9: **end if**
10: **end while**
11: **return** ϵ

Il sistema procede dunque valutando ciascun batch di candidati, ritornando un risultato se esso è stato individuato, oppure ϵ se non è stato trovato alcun

candidato (la menzione corrente è cioè valutata come *discourse new*). Sebbene si sia discusso l'utilizzo di una versione estesa del paradigma *mention-pair*, si è solo accennato al meccanismo che consente di fornire informazioni sul cluster alla rete. Per fornire una descrizione appropriata, è necessario infatti descrivere la rete neurale che è stata utilizzata per determinare se due menzioni siano coreferenti, non prima di fornire una panoramica sullo sviluppo e su altri modelli che sono stati implementati.

3.4 Una breve storia dello sviluppo

Prima di procedere ad una descrizione della rete neurale utilizzata per il modello *mention-pair* modificato presentato in precedenza, che è risultato essere il più efficace fra quelli realizzati, è utile fornire una breve descrizione dello sviluppo, che ha condotto a implementare vari modelli per la *coreference resolution*, che sono tuttavia stati scartati a causa delle scarse performance che essi offrivano.

Il primo modello utilizzato è stato un modello che possiamo definire *sequence ranking*. Tale modello operava infatti una scelta binaria inizialmente fra due frasi, confrontandole con una *referring expression*, in modo da individuare quale frase venisse valutata come coreferente dal sistema. Si procedeva poi scomponendo la frase seguendo il suo albero sintattico e valutando quale sequenza testuale venisse considerata coreferente.

Algoritmo 6 Sequence Ranking model

1: **Input**: a referring expression r_e
2: $c_l \leftarrow$ the phrases before the referring expression and all nodes of the syntax tree that appear before the refferring expression in the same phrase
3: **while** $|c_l| > 0$ **do**
4: **while** $|c_l| = 1$ **do**
5: $c \leftarrow \text{popleft}(c_l)$
6: **if** c was already subdivided **then**

```
7:              return c
8:          else
9:              get all children of c in his syntactic tree, add them to c_l
10:             append(c,c_l)
11:         end if
12:     end while
13:     c_1 ← popleft(c_l)
14:     c_2 ← popleft(c_l)
15:     r ← select_coreferent(c_1,c_2,r_e)
16:     if r ≠ ϵ then
17:         append(r,c_l)
18:     end if
19: end while
```

L'algoritmo 6 si utilizza dunque una primitiva select_coreferent, che ritorna quale dei due candidati c_1,c_2 il sistema ritenga coreferente con r_e, oppure ϵ se non ritenga che nessuno dei due sia coreferente. Tale selezione viene effettuata da una rete neurale che utilizzava LSTM sulle sequenze di *word embedding* delle parole dei candidati. Una volta che è rimasta una sola frase, essa viene scomposta finché non si sono valutati tutti i suoi figli, che a loro volta possono essere scomposti, e così via fino ad ottenere una lista di candidati vuota (in questo caso r_e è *discourse new*) oppure si è selezionato il candidato migliore. Si noti che il candidato che viene scomposto è nuovamente aggiunto alla fine della lista dei candidati, in modo da valutarlo rispetto ai nodi figli nell'albero sintattico.

Tale sistema ha però una serie di problemi che ne hanno causato l'abbandono in favore di altri modelli. Innanzitutto esso non è in grado di valutare la situazione nella quale vi sia un solo candidato che è anche una foglia dell'albero sintattico. In questo caso infatti esso non può essere scomposto e viene selezionato come antecedente, anche se la *referring expression* fosse *discourse new*. Per tale ragione in questa fase si sono considerati solo elementi non *discourse new*. Si è inoltre rilevato come nel corpus le menzioni in una catena

di coreference non corrispondano necessariamente a strutture dell'albero sintattico, anche nel caso di annotazioni manuali e quindi esatte. Un problema ancor più catastrofico riguarda però la fase iniziale dell'esecuzione, quando vengono comparate intere frasi. Un errore in questa fase, infatti, può condurre a scartare sequenze molto lunghe, che contengono tuttavia almeno una menzione coreferente alla *referring expression*. Nonostante l'uso di due livelli di LSTM, che conservano nella memoria della rete l'intera sequenza di *word embedding* in modo più efficace, tali errori sono risultati essere frequenti e hanno compromesso in modo irreparabile i risultati ottenuti. Per tale ragione si è passati ad una seconda tipologia di modello, molto simile al *mention-pair* model modificato utilizzato nella versione finale, che procediamo a descrivere brevemente.

Il secondo modello utilizzato procedeva determinando se due menzioni fossero coreferenti, sfruttando informazioni sulle entità a cui le due menzioni appartenevano, lo definiremo dunque *entity-based model*, poiché tale modello non è conforme a quelli discusso nella sezione 1.6. A differenza del modello *mention-pair* modificato, *entity-based* esamina in ordine di apparizione ciascuna menzione, alla ricerca dei suoi coreferenti nel testo successivo. Per questa ragione, non vi sono garanzie sul fatto che nessuna delle due menzioni non sia già stata valutata come parte di un'entità. Alcune informazioni sul una serie di candidati coreferenti e sulle rispettive entità vengono poi fornite ad una rete neurale, che determina se esse siano coreferenti o meno, sfruttando batch di candidati ottenuti con l'algoritmo 4, in cui però l'ordine dei candidati risulta invertito e, a differenza di quanto visto in precedenza, si utilizzavano i candidati dopo la menzione. In caso di coreferenza si creava infine una *coreference chain* composta dalla fusione delle *coreference chain* delle due menzioni.

Tale modello risultava però essere eccessivamente complesso rispetto a un modello *mention-pair* come quello descritto in precedenza, senza inoltre fornire particolari vantaggi. In fase di training della rete, oltretutto, si rendevano necessarie complesse operazioni per non dover supporre che le entità di

cui facevano parte le due menzioni venissero individuate in maniera perfetta. Si procedeva dunque aggiungendo solo alcune delle menzioni di ciascuna entità tramite un generatore di numeri casuali invocato per ciascuna menzione dell'entità. Nel caso le menzioni facessero parte della stessa entità, si popolava la *coreference chain* della menzione corrente tramite un generatore di numeri casuali, si procedeva poi in modo analogo per quanto riguarda l'entità del candidato, considerando però solo le menzioni che non erano state aggiunte all'entità della menzione corrente. Sebbene i risultati ottenuti da tale algoritmo fossero incoraggianti, esso introduceva complessità che non si traducevano in miglioramenti nelle prestazioni. Per tale ragione si è deciso di mantenere l'impianto del modello *entity-pair*, cioè l'utilizzo di informazioni sull'entità a cui appartiene una menzione, semplificando però il modello.

3.5 Struttura della rete neurale

Una volta introdotto il paradigma utilizzato dal *solver*, è necessario descrivere la struttura della rete neurale che fornisce al sistema informazioni su quali coppie antecedente-menzione siano coreferenti. A tal fine è utile procedere per gradi, introducendo dapprima il sistema in maniera astratta e descrivendo via via le componenti da cui esso è formato.

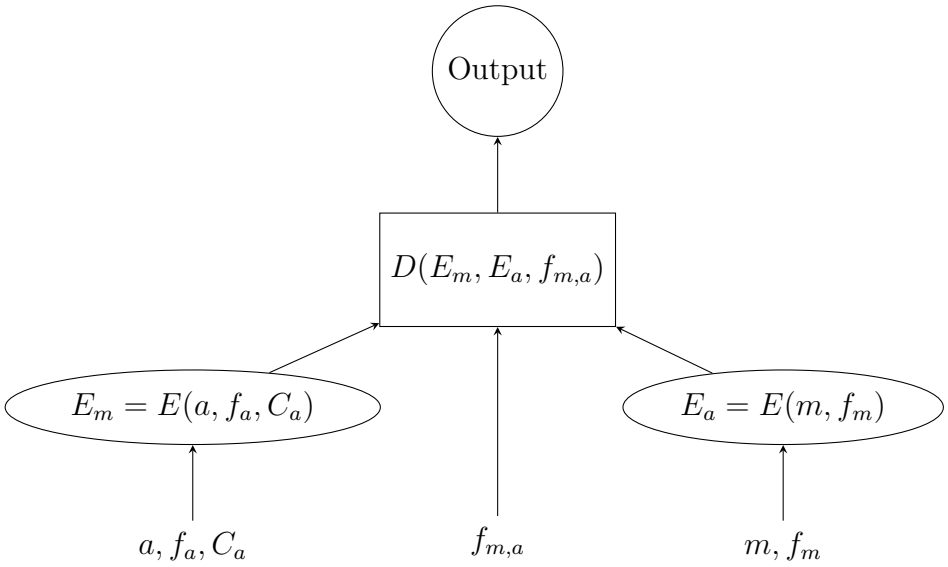


Figura 3.1: Una visione d'insieme della rete

La figura 3.1 mostra in modo sintetico quale sia la struttura generale della rete neurale implementata. Partendo dal basso, vengono forniti in input a ed m, ovvero il candidato antecedente e la menzione corrente, oltre alle loro rispettive *feature* f_a, f_m. Si noti come nel ramo della rete relativo all'antecedente venga inoltre fornito C_a, che rappresenta informazioni sull'entità di cui fa parte a. Vi sono inoltre *feature* che riguardano la coppia di menzioni in esame, che sono indicate come $f_{m,a}$

La rete è dunque composta da due rami, che utilizzano due funzioni E che operano sugli input suddetti. Tali funzioni (che sono per la verità distinte fra antecedente e menzione, ma che indichiamo con E per semplificare la notazione) hanno lo scopo di fornire una codifica della menzione e del suo potenziale antecedente, in modo che la rete possa sfruttare questa prima astrazione, che aggrega e combina *feature*, la menzione stessa (come sequenza di parole) e, nel caso dell'antecedente, informazioni sull'entità a cui esso appartiene.

Il livello successivo della rete ha infine lo scopo di combinare le rappresentazioni di menzione ed antecedente e le *feature* di relazione fra di esse, tramite una funzione che indichiamo come D. Nei test si sono utilizzate due

possibili strutture per l'output, calcolando un valore compreso fra 0 ed 1, che indica quanto la rete sia confidente che menzione corrente ed antecedente siano coreferenti, oppure utilizzando due neuroni di output con valori che rappresentano la coreferenza o meno fra le due menzioni. Tali valori risultano essere l'output della rete e vengono utilizzati per determinare se le due menzioni siano coreferenti.

Si è dunque fornita una breve panoramica delle principali componenti della rete, che verranno ora descritte in modo più esteso.

3.5.1 Codifica di menzione ed antecedente

La codifica di menzione ed antecedente è un aspetto fondamentale per quanto riguarda il nostro sistema di *coreference resolution* tramite reti neurali *deep*, in quanto è a partire da tale rappresentazione che il sistema è in grado di discriminare menzioni coreferenti e non. Da questo punto di vista l'approccio da noi utilizzato fa uso di LSTM per codificare sequenze di parole, nonché di un solo livello ReLU per combinare gli output delle LSTM e le *feature* associate alla menzione. Forniamo dunque uno schema, che illustra il meccanismo fondamentale di funzionamento dell'*encoding* della menzione, mentre descriveremo in seguito la piccola variazione utilizzata nel caso dell'antecedente.

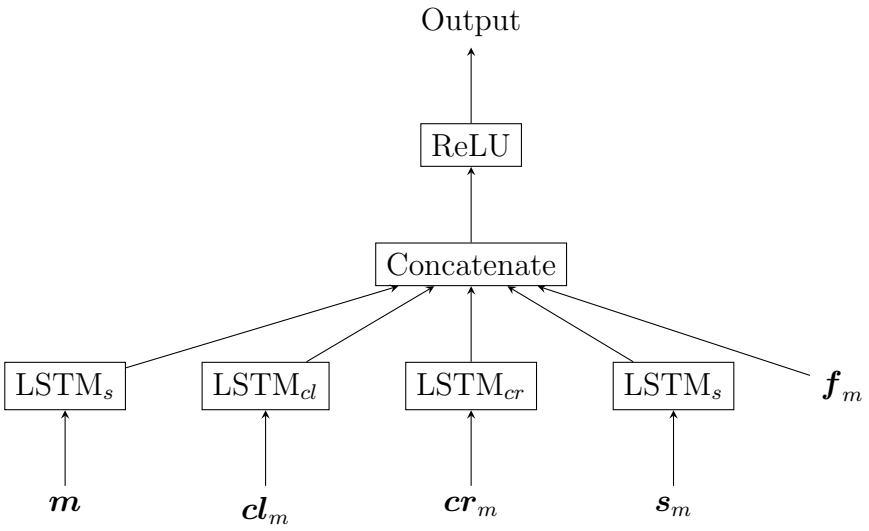


Figura 3.2: La codifica della menzione

Nella figura 3.2 partendo dal basso vengono mostrate una serie di LSTM che operano su alcune sequenze di *word embedding* ottenuti tramite word2vec. In particolare:

- $\boldsymbol{m}$: rappresenta la sequenza di *embedding* delle parole nella menzione corrente;
- $\boldsymbol{cl}_m$: il contesto sinistro della parola, ovvero fino a 5 parole alla sua sinistra che fanno parte della stessa frase della menzione;
- $\boldsymbol{cr}_m$: il contesto destro della parola;
- $\boldsymbol{s}_m$: lo speaker (se presente) della frase in cui è presente la menzione.

Si noti come ad alcune LSTM sia stato associato lo stesso pedice s (sequence), poichè esse condividono gli stessi pesi, in quanto compiono la stessa operazione di codifica di *word embedding* di menzioni, in questo caso riguardanti lo speaker e la menzione corrente, consentendo un riutilizzo dei loro pesi. Tale semplificazione consente di ridurre il numero di pesi della rete, semplificandola e consentendo dunque un apprendimento ed un testing più rapido,

nonché un minore utilizzo di memoria. Per quanto riguarda il contesto sinistro e destro si sono qui indicate due LSTM distinte, rispettivamente LSTM_{cl} e LSTM_{cr}. Durante la fase di testing, tuttavia, si sono testate due versioni della rete, che utilizzano LSTM diverse per contesto sinistro e destro, oppure la stessa LSTM per menzione, contesto sinistro e destro, speaker.

Oltre a tali valori in input, codificati da sequenze di vettori di word *embedding*, sono state utilizzate una serie di *feature* $\boldsymbol{f}_m$ della menzione. Tali *feature* riguardano in particolare genere, numero, *animacy*, tipo di menzione, tipo di named entity, persona della menzione. Sono codificate tramite vettori *one-hot* della dimensione specifica per ciascuna singola *feature*.

Le *feature* e gli output delle LSTM vengono poi concatenati, per essere successivamente forniti in input ad un singolo livello ReLU, che ha lo scopo di combinare tali valori e fornire un *encoding* per la menzione.

Si procede dunque alla descrizione del funzionamento della codifica dell'antecedente. Si ricordi come si sia indicata la volontà di utilizzare informazioni sull'entità a cui l'antecedente appartiene, da cui si ottiene la seguente struttura di rete:

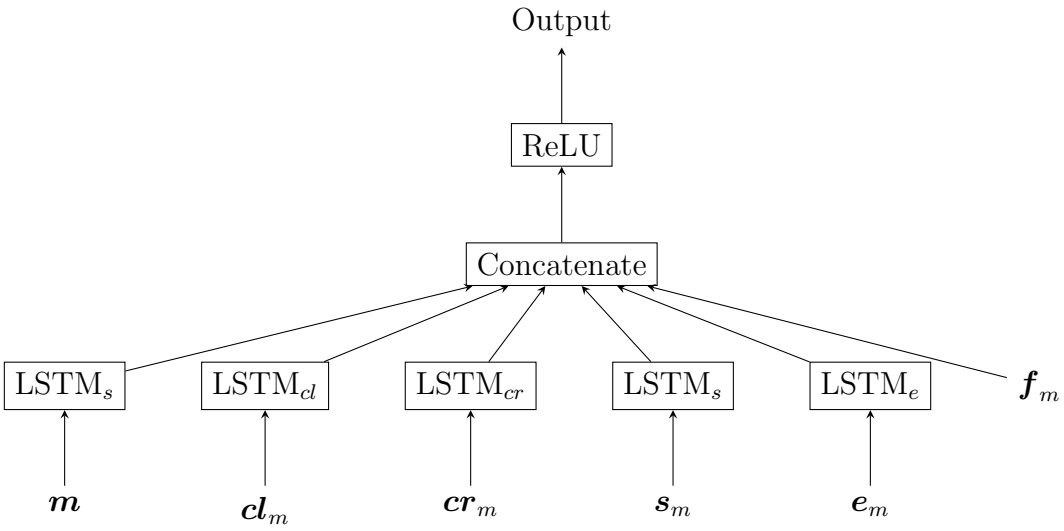


Figura 3.3: La codifica dell'antecedente

Si noti esaminando la figura 3.3 come la struttura della rete non sia molto differente e come i pesi delle LSTM_s, LSTM_{cl}, LSTM_{cr} siano condivisi con il ramo della rete che codifica la menzione corrente. Vi è tuttavia un'ulteriore input e_m, che rappresenta una sequenza *word embedding*, ottenuti dalle menzioni che sono parte della stessa entità di cui fa parte l'antecedente. Tali menzioni vengono poi separate dal *word embedding* ottenuto dal carattere ";". Tale approccio consente in maniera immediata di sfruttare informazioni sull'entità di cui fa parte l'antecedente senza ricorrere a strutture di rete più complesse. Si noti come si sia utilizzata una LSTM distinta, poiché essa deve essere in grado di distinguere ";" come carattere di separazione fra menzioni. LSTM_e, a differenza di LSTM_s, opera su sequenze di *word embedding* che rappresentano menzioni diverse e non contigue nel documento. Si tenga tuttavia presente che, a causa di vincoli temporali, non si sono effettuate analisi conclusive sull'efficacia di tale approccio.

Si sono dunque descritti i rami della rete che forniscono la codifica di menzione ed antecedente, si procede ora con la descrizione della restante porzione della rete, che ha lo scopo di combinare le codifiche e determinare se la menzione corrente e l'antecedente siano coreferenti.

3.5.2 Combinare le codifiche e determinare se le menzioni sono coreferenti

Una volta ottenute le codifiche della menzione e dell'antecedente, la rete deve combinarle per ottenere nel neurone di output un valore *floating point* che misuri la confidenza della rete nel supporre che le due menzioni siano coreferenti. A tale scopo si utilizza una struttura di rete basata sull'utilizzo di ReLU, che combina le due codifiche e le *feature* di relazione fra menzione ed antecedente.

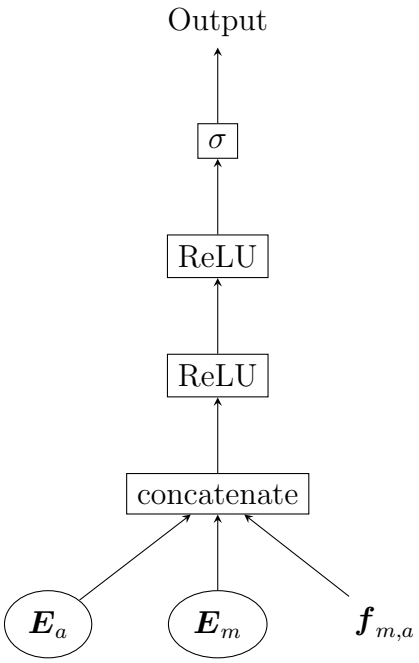


Figura 3.4: La struttura della rimanente porzione della rete nella versione binaria

La figura 3.4 mostra la versione binaria (con un solo neurone di output) della rete, che utilizza gli *encoding* $\boldsymbol{E}_m$, $\boldsymbol{E}_a$, corrispondenti rispettivamente all'*encoding* della menzione e dell'antecedente. Si utilizzano poi una serie di *feature* $\boldsymbol{f}_{ma}$ che mettono in relazione la menzione e l'antecedente:

- **Head match**: determina se le *head* di menzione ed antecedente siano la stessa parola;
- **Relaxed Match**: utilizza una versione rilassata dello *string match* utilizzata da Clark [14], che opera determinando se i nomi propri presenti nella menzione e nell'antecedente siano gli stessi;
- **Speaker match**: determina se gli speaker delle frasi della coppia menzione-antecedente siano le stesse;
- **Antecedent speaker is mention**: determina se lo speaker dell'antecedente corrisponda alla menzione;

- **Mention speaker is antecedent**: determina se lo speaker della menzione corrisponda all'antecedente.

Gli *encoding* di menzione ed antecedente e le *feature* $\boldsymbol{f}_{ma}$ vengono poi concatenate. Successivamente si utilizzano due livelli di ReLU, infine un unico neurone di output, attivato tramite una funzione sigmoide, per ottenere un valore di output nel dominio $[0, 1]$.

Si procede ora alla descrizione della variante della rete che utilizza due neuroni di output.

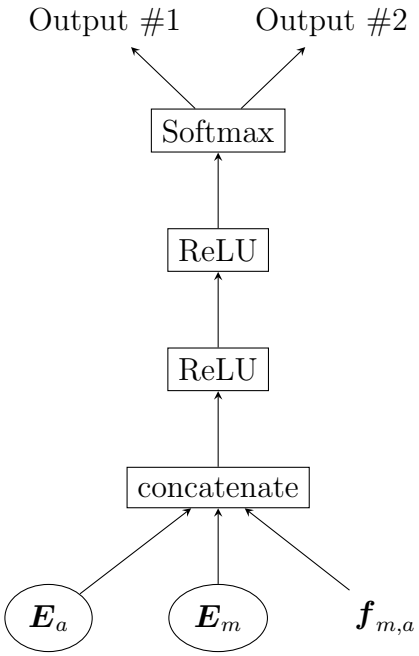


Figura 3.5: La struttura della rimanente porzione della rete nella versione con due neuroni di output

La struttura della rete mostrata in figura 3.5 è sostanzialmente invariata rispetto alla versione binaria, tuttavia gli output della rete sono due ed essi utilizzano come funzione di attivazione softmax, in modo che la somma dei loro valori sia unitaria.

Una volta fornita una descrizione della rete neurale utilizzata, ci si occuperà della descrizione dei meccanismi di training utilizzati, concludendo

infine con una descrizione dell'algoritmo utilizzato per il *testing* e quindi per la *coreference resolution* in generale.

3.6 Apprendimento

La procedura di apprendimento è una componente fondamentale del sistema di risoluzione della *coreference*, in quanto è tramite tale procedura che il sistema è in grado di apprendere come debba discriminare quali coppie antecedente-menzione siano coreferenti. È inoltre necessario valutare con attenzione la modalità di apprendimento selezionata: è possibile ridurre i tempi di apprendimento, limitando però in questo modo anche il numero di esempi disponibili e le performance della rete, oppure aumentare il numero di esempi incrementando tuttavia i tempi di calcolo richiesti.

A causa del tempo e delle risorse computazionali disponibili, si è scelto di eseguire l'algoritmo di apprendimento a partire da *golden mention*, in modo da ottenere un numero minore di candidati. Si popola dunque la lista di batch di cui si è discusso nell'algoritmo 4 con menzioni che sono contenute in fasi uguali o successive alla prima menzione della *coreference chain* a cui appartiene la menzione corrente.

A causa del grande numero di coppie menzione-antecedente che è possibile considerare (quadratico nel numero di menzioni del documento), non è stato possibile memorizzare tutte le coppie del *training set* sulle quali eseguire l'apprendimento. È pertanto necessario generare le coppie a tempo di esecuzione, operazione che richiede risorse computazionali considerevoli. Per tale ragione si è implementato un generatore di batch su cui far apprendere la rete, che parallelizza la generazione degli input e delle *label* a partire dal *training set*.

Algoritmo 7 Generatore di coppie input/valore per il training della rete neurale

1: **Input**: C, the corpus, a list of documents
2: **Input**: *batch_size*, the desired size for minibatch gradient descent

RNGseed(e)
shuffle(C)
$t \leftarrow []$
while $i < |C|$ **do**
    $b \leftarrow$ a list of batches of coreference chains in the next 300 documents
    while $|b| > 0$ **do**
        $c \leftarrow$popfirst(b)
        $R \leftarrow$ parallel_map(get_training_couples,R)
        for all $r \in R$ **do**
            concatenate(r,t)
            if $|t| \geq 30 * batch\_size$ **then**
                shuffle(t)
                while $|t| \geq batch\_size$ **do**
                    $curr \leftarrow$ remove the first $batch\_size$ elements from t
                    yield $curr$
                end while
            end if
        end for
    end while
    $i \leftarrow \max(i + 300, |C|)$
    save i and the network weights to disk
end while
while $|t| \geq batch\_size$ **do**
    shuffle(t)
    $curr \leftarrow$ remove the first $batch\_size$ elements from t
    yield $curr$
end while

Il generatore (algoritmo 7) opera dunque su batch di *coreference chain* estratti da 300 documenti per ciascuna iterazione. Si noti come venga inizializzato il *seed* del RNG tramite RNGseed(e), che consente di riprendere il training anche se vengono mescolati i documenti tramite la funzione shuffle, in quanto

per ciascuna *epoch* si ottiene lo stesso ordine di mescolamento.

Il generatore procede poi utilizzando una versione parallela della primitiva Map, parallel_map, per ottenere una lista di liste di coppie *input,label* ottenute dalle *coreference chain* fornite alla funzione get_training_couples. Nella nostra implementazione viene però utilizzata una versione asincrona di map, che ritorna le coppie *input,label* man mano che esse vengono costruite da un pool di processi. Nella descrizione dell'algoritmo si è scelto di non evidenziare tale aspetto per semplificarne la notazione. Ciascuna coppia ottenuta viene poi concatenata a quelle precedenti in modo da rispettare le relative dimensionalità, tramite la funzione *concatenate*. Si tenga presente che nell'implementazione concreta è stato necessario effettuare il *padding* delle sequenze di lunghezza variabile, in modo da consentire al sistema il training tramite batch, che richiede che esse abbiano la stessa dimensionalità per poterle utilizzare come input. Si è inoltre aggiunto un livello di *masking* alla rete neurale, che le consente di ignorare il *padding* (non vengono considerati i word vector che abbiano solo valori pari a 0).

Successivamente si attende che il sistema abbia ottenuto un numero sufficiente di coppie *input/label* (nell'esempio 30 volte la dimensione del batch size) per consentire di mescolarle ulteriormente, in modo da ottenere batch rappresentativi di più *coreference chain* possibile.

Si esegue tale operazione finché non si sono esauriti i batch dei documenti correnti, procedendo a questo punto al salvataggio dei dati necessari alla ripresa del training nel caso di interruzioni inattese (l'indice del prossimo documento da processare e i pesi della rete neurale).

Una volta esauriti tutti i documenti si controlla infine se sia possibile fornire alla rete un ulteriore batch (se, in sostanza, il numero di coppie rimanenti è superiore a $batch\_size$ ma inferiore a $30 * batch\_size$), altrimenti la funzione ritorna, causando l'inizio di una nuova *epoch*.

Si è dunque fornita una descrizione del generatore di esempi per il training, ma non si è definita la funzione *get_training_couples* utilizzata dalla *parallel_map*. Essa risulta essere analoga a quanto visto in precedenza

durante la descrizione del modello *mention-pair* da noi modificato.

Algoritmo 8 Costruttore di esempi per il training

```
1: Input: the coreference chain c
2: r ← []
3: for all m ∈ c do
4:     l_m ←get_candidates(m)
5:     while |l_m| > 0 do
6:         b ←popfirst(l_m)
7:         for all a ∈ b do
8:             e ← (m, a, is_coreferent(m,a))
9:             append(e,r)
10:        end for
11:    end while
12: end for
13: return r
```

Le coppie input/label per una data *coreference chain* vengono dunque individuate a partire dalla lista di candidati costruita tramite l'algoritmo 4, in modo da rispecchiare il funzionamento del *solver* durante l'esecuzione vera e propria. Il numero di candidati viene tuttavia limitato, come detto, considerando solo le menzioni che non precedono la prima menzione della *coreference chain* in esame. Se viene individuato un coreferente, a differenza di quanto avviene in fase di testing, si procede considerando tutti gli altri candidati, in modo da aumentare il numero di esempi e consentire alla rete di distinguere anche candidati che non sono immediatamente precedenti alla menzione corrente. Nel caso non vi siano antecedenti della menzione corrente vengono esaminati tutti i candidati fino all'inizio del documento.

Una volta descritto il generatore e la funzione utilizzata per ottenere esempi a partire da ciascuna *coreference chain*, è necessario descrivere le primitive di apprendimento utilizzate. L'utilizzo di un generatore ha consentito infatti di sfruttare una primitiva della libreria keras chiamata *fit_generator*.

Tale primitiva permette l'utilizzo di un generatore di esempi di training per evitare un utilizzo insostenibile di memoria durante l'esecuzione. Se le specifiche richiedono che il generatore fornisca dati in modo continuo senza mai ritornare, nel *solver* implementato è risultato utile sfruttare in modo non ortodosso tale primitiva per generare tutti gli esempi di una singola *epoch* di apprendimento. A tal fine si è fatto ritornare il generatore, causando un'eccezione che viene poi gestita, salvando inoltre lo stato della rete dopo ciascuna *epoch*. Tale approccio, unito alla memorizzazione su disco dell'*epoch* corrente e del prossimo documento su cui si debba fare training, consente di riprendere manualmente l'apprendimento in caso di interruzioni inattese.

Rispetto alla funzione *loss* utilizzata in fase di apprendimento, si è scelta una variante della *categorical cross entropy*, che viene fornita da keras, detta *binary cross entropy* per quanto riguarda la versione della rete con un solo neurone di input. La ragione di tale scelta risiede nella possibilità, fornita da tale funzione, di utilizzare un solo neurone di output per rappresentare una scelta binaria, che nel nostro caso corrisponde alla presenza di una relazione di coreferenza fra la menzione corrente e l'antecedente. Si è invece utilizzatal la *categorical cross entropy* per la versione con due neuroni di output. Per quanto riguarda l'ottimizzatore, si è utilizzato Adam, che, a differenza di *minibatch gradient descent*, è in grado di utilizzare informazioni statistiche sugli aggiornamenti applicati alla rete nei batch precedenti per calcolare il passo di aggiornamento successivo. Sfortunatamente non è stato possibile valutare in maniera approfondita ottimizzatori alternativi, quali ad esempio rmsprop.

Si è dunque discusso l'approccio utilizzato per il training della rete neurale, fornendo sia i dettagli implementativi riguardanti l'utilizzo di un generatore di esempi, che sfrutta un approccio multiprocesso, sia aspetti legati all'algoritmo di training vero e proprio applicato alla rete. Si discuterà in seguito di alcuni aspetti legati alla validazione e al *testing* dell'algoritmo, che non sono stati trattati in precedenza.

3.7 Validazione e Testing

L'utilizzo di meccanismi di valutazione automatici (su *validation set*) durante l'apprendimento è una pratica comune nell'ambito del *machine learning* e risulta essere particolarmente utile quando si vuole evitare l'*overfitting* della rete sul *training set*. Tale possibilità, che pure è fornita dalla primitiva *fit_generator* utilizzata per l'apprendimento, non è però stata sfruttata. Il tempo di calcolo a disposizione, anche a causa dello sviluppo di varie iterazioni del *solver*, che hanno comportato modifiche sostanziali al paradigma di *coreference resolution* utilizzato, non consentivano di poter validare i risultati a *runtime*, soprattutto nel periodo iniziale, nel quale si sono testati modelli di *coreference resolution* molto dispendiosi dal punto di vista computazionale. L'utilizzo di macchine del laboratorio Ercolani, inoltre, forniva la possibilità di valutare in parallelo l'andamento dell'apprendimento in corso su un'altra macchina , in modo da ottimizzare le risorse. Si sono pertanto implementate due modalità di testing che vengono invocate tramite opzioni dello script python, che sono state invocate manualmente ma frequentemente durante lo sviluppo per misurare l'efficacia delle soluzioni implementate.

A causa delle limitazioni del modello *sequence ranking* inizialmente utilizzato, risultava indispensabile che ciascuna menzione venisse associata all'antecedente immediatamente precedente, salvo incorrere in forti penalizzazioni nelle performance del sistema. Pertanto si è costruito un sistema di *testing* rudimentale e fortemente semplificato rispetto alla risoluzione del problema reale, che fornisse informazioni dettagliate su ciascuna assegnazione di coreferenza individuata, rispetto alla *coreference chain* reale. Venivano naturalmente fornite anche informazioni aggregate, in modo da poter meglio comprendere i progressi dell'algoritmo. Tale funzionalità è stata utilizzata ampiamente per la risoluzione di *bug* e per la valutazione dell'andamento del programma in termini generici. Nelle prime iterazioni dell'algoritmo le performance risultavano comunque essere deludenti, tuttavia la possibilità di valutare rapidamente le performance tramite tale test semplificato ha consentito di individuare i benefici della corrente implementazione del *solver*.

In una seconda fase, quando l'algoritmo appariva più maturo, è stato implementato un sistema di *coreference end-to-end*, tramite il quale è stato possibile validare in modo frequente l'andamento delle performance. Tale sistema procede come si è discusso nella sezione 3.3, individuando via via gli antecedenti delle menzioni che gli vengono fornite. Per ridurre i tempi di *testing*, si sono utilizzate durante lo sviluppo menzioni *gold*, che hanno il vantaggio di essere molto meno numerose di menzioni estratte in modo automatico. L'approccio utilizzato per tali test sfrutta la procedura utilizzata per individuare l'antecedente, che chiameremo find_antecedent, descritta nell'algoritmo 5.

Algoritmo 9 Costruisci coreference chain

Input: current document d
Input: the list of mentions in d, L
$cc_l \leftarrow []$
for all $m \in L0$ **do**
    $a \leftarrow$ find_antecedent(m)
    if $\exists cc | cc \in cc_l \wedge m \in cc$ **then**
        append(a,cc)
    else
        append($[m,a]$,cc)
    end if
end for
return cc_l

Il sistema procede dunque individuando l'antecedente di ciascuna menzione, distinguendo poi fra *coreference chain* già individuate e nuove. Nel caso m appartenga già ad una *coreference chain*, infatti, si procede all'aggiunta dell'antecedente a a tale *coreference chain*. Viene altrimenti creata una nuova *coreference chain* contenente la menzione e l'antecedente. Nell'implementazione pratica viene utilizzato un dizionario, che tiene traccia di quali menzioni

appartengano a quali *coreference chain*, in modo da consentire un'esecuzione più rapida della procedura.

Una volta ottenuta una lista di *coreference chain* per un certo documento, si procede poi ad una memorizzazione su disco di una versione minimale dei documenti nel formato CoNLL, in cui vengono incluse le *coreference chain* che sono state individuate. È dunque possibile utilizzare lo script *reference coreference scorer* per la valutazione delle metriche della *coreference*, di cui daremo conto nella sezione successiva.

Un'ultima procedura di valutazione è stata implementata per quanto riguarda la qualità delle menzioni: viene utilizzata una ricerca binaria sulla lista ordinata di candidati fornita al sistema per cercare ciascuna menzione di ciascuna *coreference chain* al loro interno. Si valutano in questo modo le menzioni estratte in modo automatico, tramite la serializzazione su disco di file CoNLL e lo script di valutazione ufficiale. Tale procedura consente inoltre di ottenere un limite superiore alle performance ottenibili tramite menzioni estratte in modo automatico.

3.8 Valutazione

Poiché si è conclusa la discussione dei meccanismi utilizzati per valutare il *solver*, è ora possibile presentare le performance che si sono ottenute tramite l'ultima versione disponibile dello script "*reference coreference score*". Si valutano tre diverse versioni della rete:

- **Binary**: utilizza gli stessi pesi per ciascuna LSTM, tranne che per quanto riguarda l'LSTM che codifica le menzioni dell'entità a cui appartiene l'antecedente, un solo neurone di output, nessun *dropout*;

- **Categorical**: utilizza lo stesso tipo di LSTM di *binary*, ma due neuroni di output, *dropout* applicato agli output delle LSTM;

- **More LSTMs**: vengono utilizzate reti LSTM distinte per il contesto destro e sinistro della menzione, due neuroni di output, *dropout* applicato agli output delle LSTM.

Si presentano dunque le metriche necessarie al calcolo della f-score media di CoNLL 2012 durante le prime 10 epoch di training sul *validation set* con *golden mention*. Si procederà poi a presentare i risultati ottenuti rispetto al test set di CoNLL 2012 utilizzando la rete che ha mostrato le migliori performance, sia per quanto riguarda *golden mentions* che per quanto riguarda menzioni estratte in modo automatico.

La prima metrica valutata è MUC (1.7.2), che misura le performance in termini di accuratezza nella creazione di collegamenti fra menzioni nella coreference chain. Si valuteranno i valori di *precision*, *recall* ed *F-score* definiti dalla metrica dopo ciascuna epoch di training.

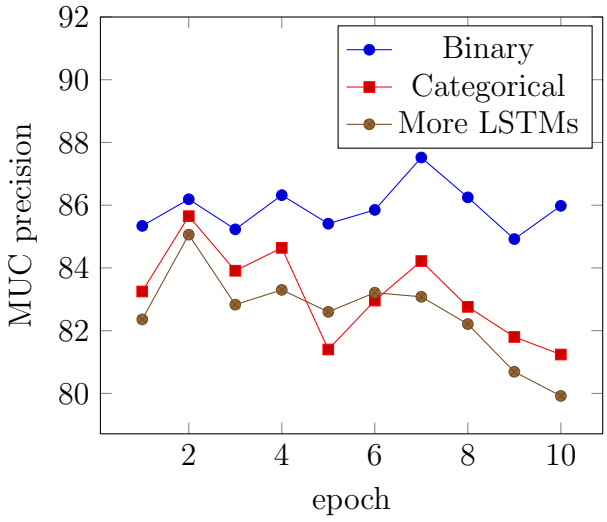


Figura 3.6: Precision MUC

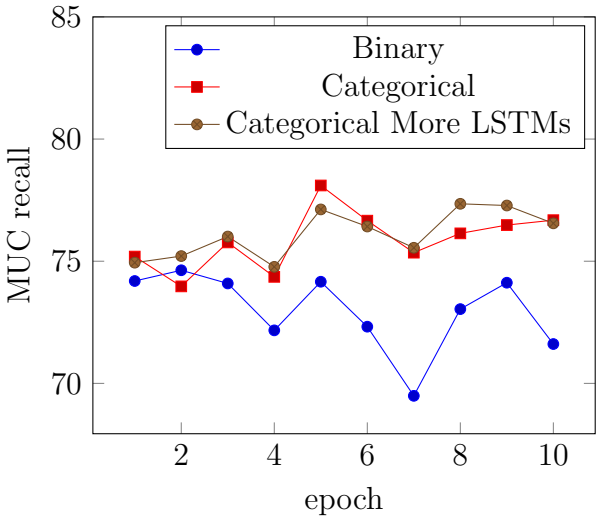


Figura 3.7: Recall MUC

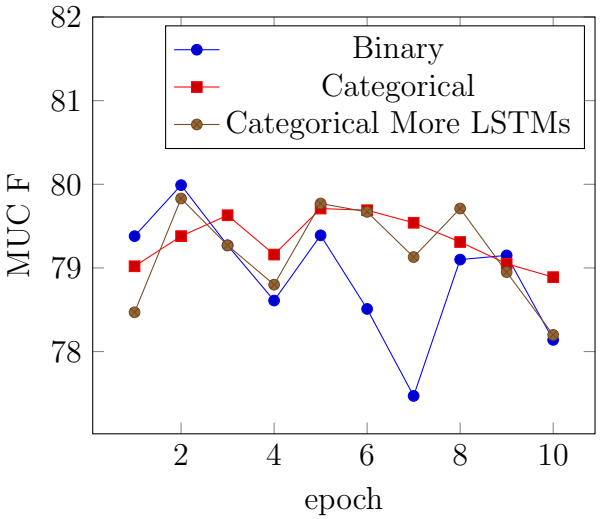


Figura 3.8: F-score MUC

Si presenta in seguito la metrica B-cubed f-score (sezione 1.7.2), per quanto riguardano i valori di *precision*, *recall* ed *F-score*.

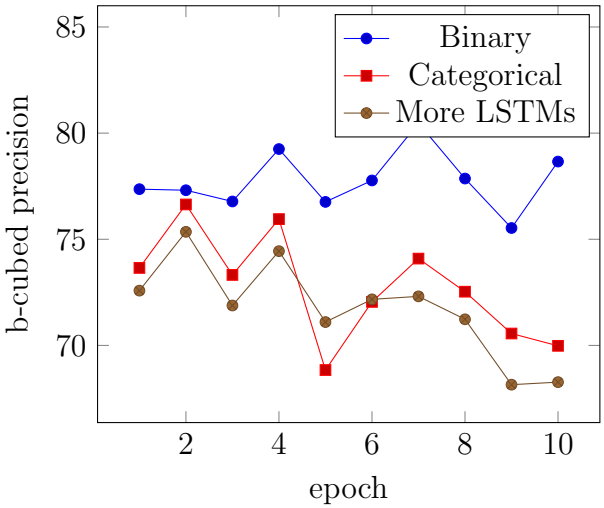


Figura 3.9: Precision B-cubed

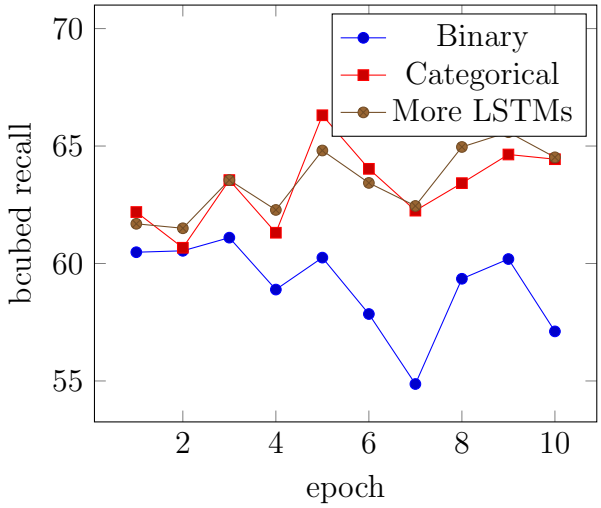


Figura 3.10: Recall B-cubed

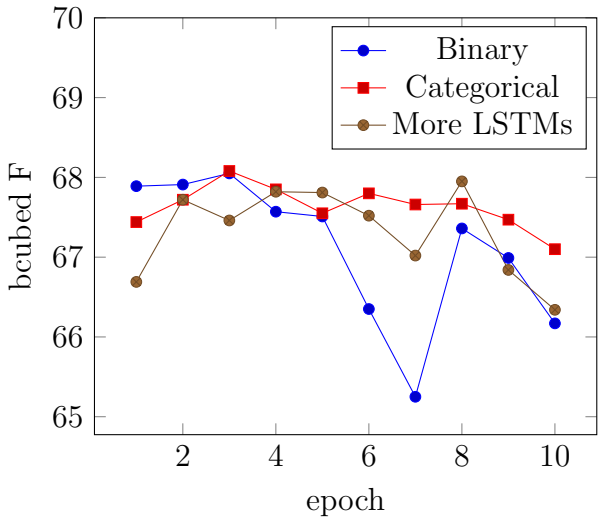


Figura 3.11: F-score B-cubed

Si preseantano poi i risultati ottenuti secondo la variante di CEAF basata sulle entità (sezione 1.7.2):

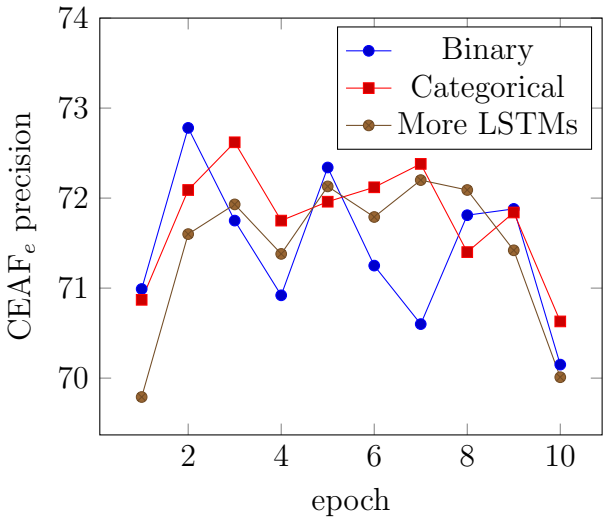


Figura 3.12: Precision CEAF$_e$

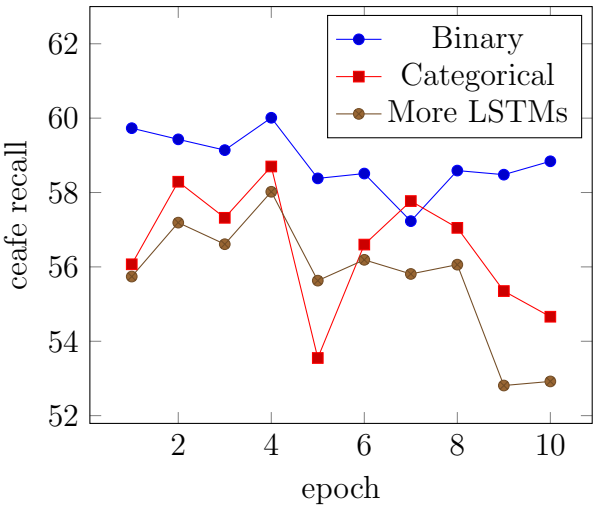


Figura 3.13: Recall CEAF$_e$

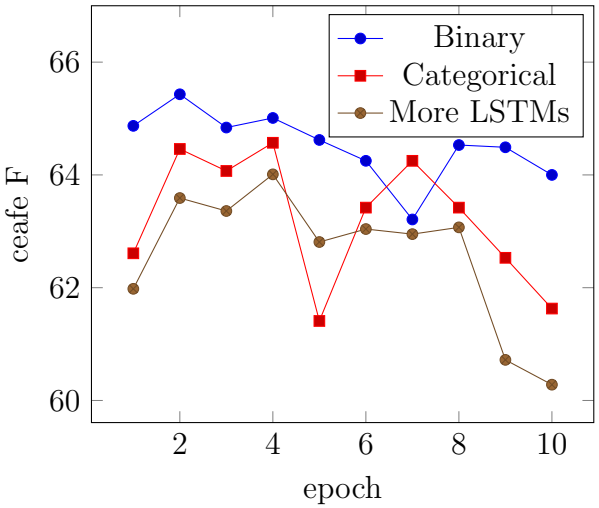


Figura 3.14: F-score CEAF_e

Si mostra infine l'andamento della metrica CoNLL 2012 (sezione 1.7.2) durante le epoch:

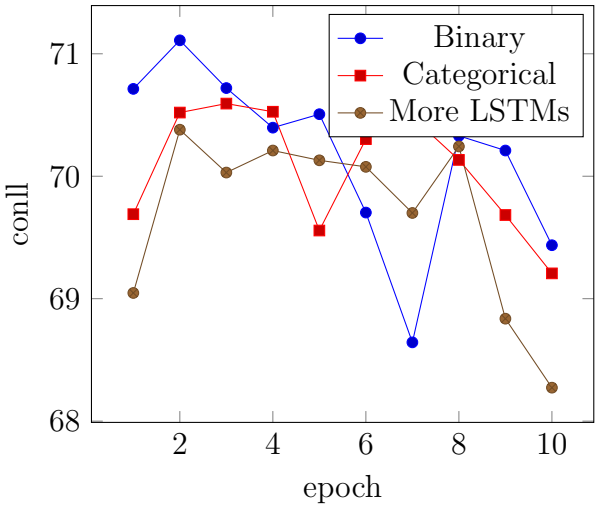


Figura 3.15: CoNLL 2012

È evidente come in tutti i test il sistema incorra in *overfitting* già dopo poche *epoch* di training. Una delle ragioni di tale comportamento è da individuarsi nella scelta di non utilizzare o utilizzare in modo limitato il "*dropout*", che è in grado di mitigare tale situazione, oltre che nell'utilizzo

di *golden mentions* durante il training, che riducono fortemente il numero di esempi disponibili per l'apprendimento. Si mostrano infine i risultati ottenuti a partire dalla rete ottenuta dopo la seconda *epoch* della versione *binary* della rete, che è risultata la più performante. Si riportano i risultati del sistema sia su golden mentions che su menzioni individuate in modo automatico, per quanto riguarda le metriche MUC, B-Cubed, $CEAF_e$ e la metrica da esse derivata, CoNLL.

	MUC			B-Cubed			$CEAF_e$			CoNLL
	Pre	Rec	F_1	Pre	Rec	F_1	Pre	Rec	F_1	Avg F_1
Gold	86.1	80	80.7	75.2	60.5	67	71.8	57.8	64	70.6
Auto	39.1	61.5	47.9	29	47	35.9	26.6	44	33.1	39

È evidente come, se le prestazioni dell'algoritmo per quanto riguarda le *golden mentions* possono essere considerate di buon livello, le performance riguardanti le menzioni estratte in modo automatico non sono competitive. Si ricordi che il *training* è stato effettuato su *golden mentions* per ridurre il tempo di apprendimento, per cui rimane da valutarsi l'utilizzo del nostro *solver* se venissero utilizzate menzioni automatiche per il *training*.

3.9 Sviluppi Futuri

Si sono dunque discusse le performance della rete sul *dataset* CoNLL 2012. È evidente come, se vi è ampio spazio di miglioramento per la rete per quanto riguarda l'utilizzo di menzioni estratte in modo automatico, le prestazioni ottenute su *golden mentions* siano confortanti. Sfortunatamente per ottenere risultati in tempi accettabili è stato necessario compiere una serie di compromessi per ridurre i tempi di training della rete, che hanno sicuramente avuto un impatto considerevole sulle performance. Si deve infatti considerare il fatto che gli approcci descritti nella sezione 2.2 richiedono svariati giorni di training su GPU molto potenti, tempi di apprendimento che sarebbero stati per noi proibitivi su CPU. Si ricordi inoltre come l'apprendimento della rete sia stato effettuato su menzioni *gold*, mentre sarebbe stata necessaria

una fase di training su menzioni estratte in modo automatico, che forniscono un numero di esempi più elevati ed avrebbero probabilmente consentito di meglio riconoscere menzioni che non sono parte del *golden standard*.

Al fine di ottenere un sistema di *coreference end-to-end* è lecito supporre che il modello *mention-pair* modificato da noi proposto non sia sufficiente. Sarebbe probabilmente necessario utilizzare un'ulteriore rete neurale per effettuare il *ranking* delle menzioni, in modo da considerare solo un numero limitato di potenziali antecedenti per ciascuna menzione, riducendo dunque la possibilità di errori, in modo analogo a quanto proposto da Clark e Manning (si veda la sezione 2.2.1)

Un ulteriore aspetto riguarda la possibilità di restringere il numero di antecedenti tramite regole statiche, considerando ad esempio l'accordo dei generi fra le due menzioni come un filtro di candidati. Tale approccio richiede tuttavia approfondite analisi sul fenomeno della *coreference* e sulla sua annotazione nel corpus da noi esaminato, senza la quale si potrebbe incorrere in errori gravi, peggiorando ulteriormente le performance del sistema.

Le dimensioni della rete utilizzata sono state ridotte in modo drastico rispetto alle implementazioni discusse nella sezione 2.2, per cui sarebbe necessario valutare l'utilizzo di una rete di dimensioni più ampie. In particolare, sarebbe possibile utilizzare contesti di dimensioni diverse nell'ambito della codifica delle menzioni, in modo analogo a quanto visto per l'algoritmo di Clark nella sezione 2.2.1. Nella stessa discussione dell'algoritmo vengono descritte altre *feature* da noi non sfruttate, che potrebbero essere aggiunte alla rete per migliorarne le prestazioni. La dimensione delle componenti della rete, inoltre, non è stata ottimizzata, in quanto si desiderava ottenere risultati in poche ore. L'utilizzo di vettori di output con dimensioni maggiori, in particolare per quanto riguarda gli output delle LSTM e per la rappresentazione delle menzioni, potrebbe condurre a prestazioni migliori.

Un'altra possibilità riguarda l'utilizzo di due livelli LSTM per elaborare ciascuna sequenza testuale, in particolar modo per quanto riguarda la rappresentazione del cluster di menzioni, effettuata ora tramite un solo livello

LSTM. L'utilizzo di più livelli LSTM per le sequenze consentirebbe alla rete di fornire in output più informazioni in particolare sull'entità, migliorando probabilmente le performance. In alternativa o in aggiunta a tale modifica sarebbe possibile utilizzare reti LSTM bidirezionali, che elaborano le sequenze di *word embedding* sia in ordine di apparizione, sia in ordine inverso, concatenando poi i due risultati così ottenuti.

Al fine di ottenere risultati in tempi utili, si è scelto inoltre di utilizzare dropout sui soli output delle LSTM. Tale tecnica, che annulla una percentuale prefissata di segnali nella rete, consente di ridurre il fenomeno dell'*overfitting*, aumentando tuttavia il tempo necessario per l'apprendimento. L'utilizzo di tale tecnica consente infatti alla rete di sfruttare sotto-reti ridondanti per fornire lo stesso output, caratteristica che migliora le performance della rete dati ad essa sconosciuti. Sarebbe auspicabile utilizzare più dropout su livelli diversi della rete, nel tentativo di ridurre l'*overfitting*.

Un ulteriore miglioria, che non è stata effettuata a causa di limitazioni legate all'utilizzo di h5py e TensorFlow da parte di un processo e suoi figli creati tramite *fork*, riguarda una implementazione parallela dell'algoritmo di test, che utilizzi tutti i core della macchina ed un processo per ciascun documento del corpus.

Nonostante le evidenti problematiche della rete presentata, essa costituisce una *proof-of-concept* interessante per l'utilizzo di un modello *mention-pair* con lo scopo di effettuare la scelta finale nell'ambito della *coreference*, che potrebbe ulteriormente beneficiare di una selezione iniziale di un numero di candidati ridotto rispetto a quello utilizzato in fase di *testing*.

Conclusioni

Si è dunque conclusa la discussione del *task* della *coreference resolution* e di alcuni approcci presenti in letteratura per la risoluzione di tale problema, che riguarda l'individuazione di tutte le menzioni di un testo che si riferiscono ad una stessa entità. Si è poi presentata una proposta di risolutore, che utilizza reti neurali *deep* per implementare un modello di risoluzione della coreference basato sul *mention pair model* presente in letteratura, che determina se una menzione corrente ed un suo potenziale antecedente siano coreferenti. Il modello *mention pair* così modificato sfrutta inoltre informazioni sull'entità dell'antecedente, pur non discostandosi dalla procedura di comparazione fra due menzioni.

I risultati ottenuti tramite tale solver sono stati molto positivi per quanto riguarda l'utilizzo di *golden mention*, utilizzando cioè come candidati solo le menzioni che appartengono ad una *coreference chain* presente nel testo. La fase di apprendimento della rete ha infatti utilizzato menzioni *gold*, in modo da ridurre i tempi necessari all'ottenimento di risultati. Anche le dimensioni della rete sono state limitate per quanto possibile, in modo da ridurre i tempi di apprendimento. Tali compromessi hanno però comportato sia performance non competitive per quanto riguarda l'utilizzo di menzioni estratte in modo automatico, sia un rapido *overfitting* della rete rispetto al *training set*. L'utilizzo di *training set* basati su menzioni estratte in modo automatico è stata purtroppo imposta da vincoli temporali e limitazioni nella disponibilità di hardware per effettuare i calcoli, ma è comunque lecito chiedersi se un tale approccio non avrebbe consentito di migliorare le prestazioni del sistema in

modo sensibile.

La possibilità di utilizzare un tale meccanismo in congiunzione ad altri algoritmi di tipo *mention ranking* o *cluster ranking* per ridurre il numero di candidati da fornire alla rete potrebbe inoltre condurre a prestazioni sensibilmente migliori, anche senza procedere ad un *re-training* della rete.

Nonostante queste limitazioni, il solver proposto risulta essere un interessante *proof of concept* per quanto riguarda l'utilizzo di sequenze di *word embedding* ed reti neurali *recurrent* per risolvere il problema della *coreference*.

Bibliografia

[1] Towards robust linguistic analysis using ontonotes. http://cemantix.org/data/ontonotes.html. Accessed: 2017-09-20.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[3] Chinatsu Aone and Scott William Bennett. Evaluating automated and manual acquisition of anaphora resolution strategies. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 122–129. Association for Computational Linguistics, 1995.

[4] Douglas E Appelt, Jerry R Hobbs, John Bear, David Israel, and Mabry Tyson. Fastus: A finite-state processor for information extraction from real-world text. In *IJCAI*, volume 93, pages 1172–1178, 1993.

[5] Jennifer E Arnold, Janet G Eisenband, Sarah Brown-Schmidt, and John C Trueswell. The rapid use of gender information: Evidence of the time course of pronoun resolution from eyetracking. *Cognition*, 76(1):B13–B26, 2000.

[6] Amit Bagga and Breck Baldwin. Algorithms for scoring coreference chains. In *The first international conference on language resources and evaluation workshop on linguistics coreference*, volume 1, pages 563–566. Granada, Spain, 1998.

[7] Breck Baldwin. Cogniac: A high precision pronoun resolution engine. In *Proceedings of the ACL 97 EACL 97 Workshop on Operational Factors in Practical, Robust Anaphora Resolution*, pages 38—-45. Association for Computational Linguistics, 1996.

[8] Eric Bengtson and Dan Roth. Understanding the value of features for coreference resolution. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 294–303. Association for Computational Linguistics, 2008.

[9] Susan E Brennan, Marilyn W Friedman, and Carl J Pollard. A centering approach to pronouns. In *Proceedings of the 25th annual meeting on Association for Computational Linguistics*, pages 155–162. Association for Computational Linguistics, 1987.

[10] Donald Eric Broadbent. *In defence of empirical psychology.* Methuen, 1973.

[11] David Carter. *Interpreting Anaphors in Natural Language Texts.* Halsted Press, New York, NY, USA, 1987.

[12] Eugene Charniak. *Toward a model of children's story comprehension.* PhD thesis, Massachusetts Institute of Technology, 1972.

[13] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[14] Kevin Clark and Christopher D Manning. Improving coreference resolution by learning entity-level distributed representations. *arXiv preprint arXiv:1606.01323*, 2016.

[15] Andrew Collette. Hdf5 for python. `http://www.h5py.org/`, 2008. Accessed: 2017-09-20.

[16] Dennis Connolly, John D Burger, and David S Day. A machine learning approach to anaphoric reference. In *Proceedings of International*

Conference on New Methods in Language Processing, pages 255–261, 1994.

[17] Dennis Connolly, John D Burger, and David S Day. A machine learning approach to anaphoric reference. In *New Methods in Language Processing*, pages 133–144, 1997.

[18] Rosalind A Crawley, Rosemary J Stevenson, and David Kleinman. The use of heuristic strategies in the interpretation of pronouns. *Journal of Psycholinguistic Research*, 19(4):245–264, 1990.

[19] Aron Culotta, Michael L Wick, and Andrew McCallum. First-order probabilistic models for coreference resolution. In *HLT-NAACL*, pages 81–88, 2007.

[20] Pascal Denis and Jason Baldridge. Global joint models for coreference resolution and named entity classification. *Procesamiento del Lenguaje Natural*, 42, 2009.

[21] Pascal Denis, Jason Baldridge, et al. Joint determination of anaphoricity and coreference resolution using integer programming. In *HLT-NAACL*, pages 236–243, 2007.

[22] Yoav Freund and Robert E Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

[23] Robert Gaizauskas, Kevin Humphreys, Hamish Cunningham, and Yorick Wilks. University of sheffield: description of the lasie system as used for muc-6. In *Proceedings of the 6th conference on Message understanding*, pages 207–220. Association for Computational Linguistics, 1995.

[24] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[25] Barbara J Grosz, Scott Weinstein, and Aravind K Joshi. Centering: A framework for modeling the local coherence of discourse. *Computational linguistics*, 21(2):203–225, 1995.

[26] The HDF Group. Hierarchical Data Format, version 5. http://www.hdfgroup.org/HDF5/, 1997-NNNN. Accessed: 2017-09-20.

[27] Jerry R Hobbs. Pronoun resolution. Technical report, Department of Computer Sciences, City College, City University of New York, 1976.

[28] Jerry R Hobbs. Resolving pronoun references. *Lingua*, 44(4):311–338, 1978.

[29] Jerry R Hobbs, Mark Stickel, Paul Martin, and Douglas Edwards. Interpretation as abduction. In *Proceedings of the 26th annual meeting on Association for Computational Linguistics*, pages 95–103. Association for Computational Linguistics, 1988.

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

[31] Kevin Humphreys, Robert Gaizauskas, Saliha Azzam, Chris Huyck, Brian Mitchell, Hamish Cunningham, and Yorick Wilks. University of sheffield: Description of the lasie-ii system as used for muc-7. In *Proceedings of the Seventh Message Understanding Conferences (MUC-7)*, 1998.

[32] Andrew Kehler, Laura Kertz, Hannah Rohde, and Jeffrey L Elman. Coherence and coreference revisited. *Journal of semantics*, 25(1):1–44, 2008.

[33] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[34] Harold W Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 2(1-2):83–97, 1955.

[35] Ronald W Langacker. *On pronominalization and the chain of command*. University of California, 1966.

[36] Shalom Lappin and Herbert J Leass. An algorithm for pronominal anaphora resolution. *Computational linguistics*, 20(4):535–561, 1994.

[37] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[38] Heeyoung Lee, Angel Chang, Yves Peirsman, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Computational Linguistics*, 39(4):885–916, 2013.

[39] Xiaoqiang Luo. On coreference resolution performance metrics. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 25–32. Association for Computational Linguistics, 2005.

[40] Xiaoqiang Luo, Abe Ittycheriah, Hongyan Jing, Nanda Kambhatla, and Salim Roukos. A mention-synchronous coreference resolution algorithm based on the bell tree. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 135. Association for Computational Linguistics, 2004.

[41] Xiaoqiang Luo, Sameer Pradhan, Marta Recasens, and Eduard Hovy. An extension of blanc to system mentions. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 24–29, 2014.

[42] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *ACL (System Demonstrations)*, pages 55–60, 2014.

[43] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[44] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[45] Ruslan Mitkov. Robust pronoun resolution with limited knowledge. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 2*, pages 869–875. Association for Computational Linguistics, 1998.

[46] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[47] Massimo Poesio. The mate/gnome proposals for anaphoric annotation, revisited. In *SIGDIAL Workshop*, pages 154–162, 2004.

[48] Massimo Poesio, Roland Stuckardt, and Yannick Versley. *Anaphora Resolution Algorithms, Resources, and Applications*. Springer Berlin Heidelberg, 2016.

[49] Massimo Poesio, Renata Vieira, and Simone Teufel. Resolving bridging descriptions in unrestricted texts. In *Proceedings of the Workshop on Operational Factors In Practical, Robust, Anaphora Resolution for Unrestricted Texts*, pages 1–6.

[50] Sameer Pradhan, Xiaoqiang Luo, Marta Recasens, Eduard Hovy, Vincent Ng, and Michael Strube. Scoring coreference partitions of predicted mentions: A reference implementation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 30–35, 2014.

[51] Karthik Raghunathan, Heeyoung Lee, Sudarshan Rangarajan, Nathanael Chambers, Mihai Surdeanu, Dan Jurafsky, and Christopher Manning. A multi-pass sieve for coreference resolution. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 492–501. Association for Computational Linguistics, 2010.

[52] Altaf Rahman and Vincent Ng. Supervised models for coreference resolution. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 968–977. Association for Computational Linguistics, 2009.

[53] Marta Recasens and Eduard Hovy. Blanc: Implementing the rand index for coreference evaluation. *Natural Language Engineering*, 17(4):485–510, 2011.

[54] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, pages 693–695, 1995.

[55] Candace Lee Sidner. Towards a computational theory of definite anaphora comprehension in english discourse. Technical report, Massachusetts Inst of Tech Cambridge Artificial Intelligence lab, 1979.

[56] Joel R Tetreault. A corpus-based evaluation of centering and pronoun resolution. *Computational Linguistics*, 27(4):507–520, 2001.

[57] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, 2012.

[58] Marc Vilain, John Burger, John Aberdeen, Dennis Connolly, and Lynette Hirschman. A model-theoretic coreference scoring scheme. In *Proceedings of the 6th conference on Message understanding*, pages 45–52. Association for Computational Linguistics, 1995.

[59] Paul John Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *Doctoral Dissertation, Applied Mathematics, Harvard University, MA*, 1974.

[60] Y Wilks. Readings in natural language processing. chapter An Intelligent Analyzer and Understander of English, pages 193–203. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1986.

[61] Yorick Wilks. An intelligent analyzer and understander of english. *Communications of the ACM*, 18(5):264–274, 1975.

[62] Yorick Wilks. A preferential, pattern-seeking, semantics for natural language inference. *Artificial intelligence*, 6(1):53–74, 1975.

[63] Sam Wiseman, Alexander M Rush, and Stuart M Shieber. Learning global features for coreference resolution. *arXiv preprint arXiv:1604.03035*, 2016.

[64] Sam Joshua Wiseman, Alexander Matthew Rush, Stuart Merrill Shieber, and Jason Weston. Learning anaphoricity and antecedent ranking features for coreference resolution. Association for Computational Linguistics, 2015.

[65] Xiaofeng Yangy, Jian Su, Guodong Zhou, and Chew Lim Tan. An np-cluster based approach to coreference resolution. In *Proceedings of the 20th international conference on Computational Linguistics*, page 226. Association for Computational Linguistics, 2004.

[66] Shanheng Zhao and Hwee Tou Ng. Identification and resolution of chinese zero pronouns: A machine learning approach. In *EMNLP-CoNLL*, volume 2007, pages 541–550, 2007.