Università degli Studi di Ferrara

Corso di Laurea Magistrale in Intelligenza Artificiale, Data Science e Big Data

Natural Language Processing E Topic Modeling

Named Entity Recognition: Dall'approccio Classico a quello Neurale

A.A. 2024 - 2025

Gabrieli Andrea

Indice

Introduzione

Il riconoscimento delle entità nominate (Named Entity Recognition - NER) è una delle attività principali nel processamento del linguaggio naturale (NLP). Nel periodo in cui i documenti, i testi visibili sul web aumentano giorno dopo giorno, risulta particolarmente evidente equipaggiarsi di uno strumento che sia in grado estrarre informazioni da questi contenuti.
Il Named Entity Recognition permette di identificare le entità all'interno delle frasi o dei testi. Elabora il linguaggio naturale e ne estrae le entità. Il linguaggio naturale è però spontaneo, ambiguo e modellarlo con strutture deterministiche può risultare complicato.

I primi modelli che sono stati utilizzati sono quelli probabilistici come gli HMM intorno agli anni '90. Hanno permesso di formulare il problema in linguaggio matematico. Ma utilizzando importanti semplificazioni non riuscivano a catturare l'intera variabilità del linguaggio naturale.

Successivamente con l'avvento delle reti neurali, sono state sviluppate soluzione più complesse, come le reti ricorrenti (RNN). Hanno rappresentato un importante passo avanti, permettendo la gestione del contesto di ciascuna parola.

L'obiettivo di questo approfondimento è analizzare questa transizione. Verrà prima studiato come un modello probabilistico come l'Hidden Markov Model (HMM) riesca a risolvere il problema. E successivamente verrà analizzata una soluzione neurale che cerchi di superare le problematiche relative al modello probabilistico.

1. NER: Significato e Applicazioni

Il Named Entity Recognition [1], abbreviato con l'acronimo NER, è quell'attività che analizzando parole, frasi o testi cerca di individuare le entità. È quell'attività che associa a ciascuna parola un'etichetta che ne indica la tipologia. Pensiamo all'attività di leggere un libro o un post sui social, noi mentre leggiamo siamo in grado di riconoscere automaticamente ogni parola letta. Riusciamo a identificare che "Chris Martin" è una persona, oppure che "Londra" è una città, se leggessimo la frase "Chris Martin è il leader dei Coldplay, famosa band di Londra". I calcolatori non hanno questa capacità intrinseca, ma con un po' di lavoro anche loro saranno in grado.
Il NER è dunque un sistema che dato un testo, come quello in Figura 1, riesce ad associare a ciascuna parola, l'entità di cui fa parte.

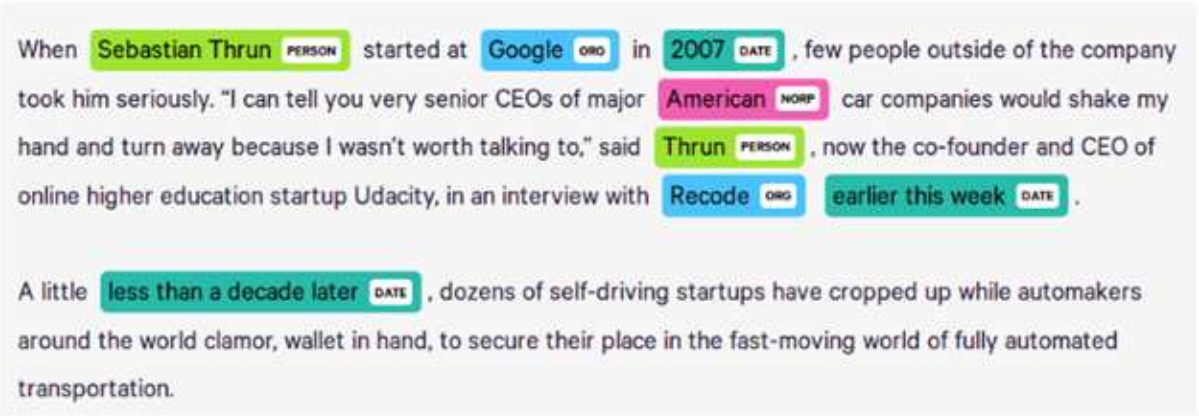

Figura 1: NER - Esempio in un estratto di testo.

Sono diverse le entità che un sistema di Named Entity Recognition è in grado di riconoscere. Alcune di queste sono:

1. Persone (PER): Si pensi a Martin Luther King, Denzel Washington, Albert Einstein.
2. Organizzazioni (ORG): Vi rientrano le aziende come Amazon, Juventus FC, oppure le istituzioni come Unione Europea, NASA.
3. Località (Entità geopolitiche, GPE): Parigi, Lido delle Nazioni, Las Vegas.
4. Quantità (QUANTITY): Sono racchiuse tutte le grandezze che possono essere misurate come 23 mg oppure 4 dL.
5. Eventi (EVENT): Usata per rappresentare particolari manifestazioni come Festival del cinema di Venezia, Olimpiadi.
6. Date: Per indicare riferimenti temporali come 14 Luglio 1789 o Natale o ancora 12/10/1492.

Riconoscere le entità non è un problema del tutto trascurabile. Infatti esistono vari casi che possono portare ad ambiguità. Per citarne uno, "Apple" può essere intesa sia come frutto, ma anche come il colosso americano. I sistemi che si intende sviluppare dovranno tenere in conto anche di questa problematica.

Vediamo ora alcuni dei campi e settori [2] in cui si utilizza il Named Entity Recognition.
Esso permette di estrarre informazioni da testi, articoli, documenti. Si possono quindi controllare i social, controllare i vari post, commenti per verificare tendenze o per aiutare aziende in strategie di marketing mirato.

Oppure nei chatbox o virtual assistance, per comprendere meglio le richieste che vengono effettuate. Ad esempio se si chiede qualcosa tipo “Come sarà il meteo dopodomani a Ferrara?” il sistema NER identificherebbe “Ferrara” come GPE e “dopodomani” come una espressione temporale, così il chatbot può rispondere in modo più preciso e dettagliato.

Oltre a questo, anche nella automatic translation, il NER può aiutare a individuare quelle entità come, ad esempio nomi propri o luoghi che non debbono essere tradotti.
Il NER viene anche utilizzato nella bioinformatica, estraendo da rapporti o documenti medici informazioni quali geni, malattie, proteine.

2. Sequence Labeling e BIO tagging

Sequence Labeling (SL) significa rispettivamente etichettatura di sequenze. È un'attività [3] in cui si identificano dei pattern ad una sequenza di token, per potergli assegnare un'etichetta. Il Named Entity Recognition è dunque un esempio di questa attività, in quanto la sequenza di token sono le varie parole che formano la frase, il testo da analizzare, e le etichette sono le entità nominate.
In prima battuta, si potrebbe pensare a una soluzione in grado di classificare ciascuna parola indipendentemente dalle altre all'interno della sequenza. Però risultati ottimali, sia nella valutazione tramite metriche come "accuracy" o "precision", sia nel scegliere l'etichetta più appropriata, sono ottenibili considerando il contesto della frase. Decidendo l'etichetta di una parola, tenendo conto di quelle già viste, o in generale di quelle a essa vicine.

2.1 Tecniche di Tagging

Per associare un'etichetta a ogni parola di una frase, è necessario definire i confini delle entità. In particolare nelle parole composte è di cruciale importanza delineare dove inizia e dove finisce ciascuna entità. I formati di tagging [4] risolvono questo problema.

Il formato di tagging più utilizzato è BIO. Questo utilizza tre etichette:

1. B (Beginning): Denota l'inizio di un'entità, viene usato per la prima parola.
2. I (Inside): Marca le parole interne di un'entità.
3. O (Outside): Indica le parole che non appartengono all'entità.

In Figura 2 è raffigurato un esempio di come questo formato etichetta una frase.

Robbie	Williams	radunò	122.000	persone	a	Knebworth	nel	1996
B-PER	I-PER	O	B-NUM	O	O	B-LOC	O	B-DATE

Figura 2: Esempio di BIO tagging.

Oltre al formato BIO esistono alcune varianti, più o meno complesse. Queste sono IO e BILUO.

Il formato IO (Inside - Outside) è la versione "ridotta". Non utilizza il tag B, e perciò indica solamente se la parola appartiene o meno all'entità. Figura 3 mostra un esempio di questa versione semplificata di tagging.

Cristiano	Ronaldo	è	nato	in	Portogallo
I-PER	I-PER	O	O	O	I-LOC

Figura 3: Esempio di IO tagging.

L’altro formato, BILUO (Beginning - Inside - Last - Unit - Outside) è la versione complessa. Utilizza 2 tag aggiuntivi rispetto BIO:

1. L (Last): Marca l’ultima parola di un’entità.
2. U (Unit): Designa le entità che sono formate da una sola parola.

In Figura 4 si può vedere come questo formato opera.

Leonardo	da	Vinci	dipinse	la	Gioconda	nel	XVI	secolo
B-PER	I-PER	L-PER	O	O	U-FAC	O	B-DATE	L-DATE

Figura 4: Esempio di BILUO tagging.

2.1.1 Quale formato scegliere?

Il formato BIO è quello più comune, quello più utilizzato. Rappresenta un ottimo compromesso quando si vuole un formato standard. Riesce bene a distinguere i confini delle entità anche quando le parole sono composte, come “Robbie Williams” nell’esempio trattato.

Il formato IO è la versione più semplice. Infatti può fare fatica a delineare i confini usando un tag in meno. Il formato BILUO, invece, è la versione complessa. Grazie ai due tag aggiuntivi denota al meglio i confini, soprattutto nelle parole singole come nell’esempio “Gioconda”.

La scelta di quale formato utilizzare dipende molto dal che cosa si vuole creare. BILUO rappresenta la scelta ideale quando si vuole realizzare un sistema di alto livello, con buoni livelli di accuracy. IO quando si vuole realizzare un prodotto semplice e quando non è di cruciale importanza l’informazione sull’inizio delle entità. In generale, qualunque sia il formato scelto, il passaggio da uno all’altro è semplice e intuitivo, ma allo stesso tempo, la scelta può influire sulle prestazioni del modello.

3. Approccio classico per il NER: HMM tagger

Per risolvere il Named Entity Recognition, la soluzione classica ricorre al Modello di Markov Nascosto. In questo capitolo verrà analizzato sia come è stato sviluppato questo modello, sia come questo venga usato per risolvere il NER.

3.1 Fondamenti di un Hidden Markov Model

Il modello di markov nascosto [5] nasce sulla base di una catena di Markov, questa è stata ideata per la prima volta dal matematico russo Andrej Andreevič Markov agli inizi del '900. Dai suoi studi, Markov verificò che la probabilità di un evento futuro dipendesse solamente da quello attuale.

Facendo un passo indietro, possiamo dire che un processo è di tipo stocastico quando esso è formato da una successione di variabili aleatorie, in cui ciascuna di esse determina lo stato del sistema in quel particolare momento.
Un processo stocastico diventa un processo markoviano quando nel prevedere lo stato futuro si fa riferimento solamente allo stato attuale e si dimenticassero tutti gli stati precedenti. Questa è l'ipotesi di Markov, viene anche rinominata assenza di memoria, e viene scritta in termini matematici con:

$$P(x_{t+1} | x_t, x_{t-1}, \ldots, x_0) = P(x_{t+1} | x_t)$$

dove x_t indica lo stato attuale e x_{t+1} lo stato successivo che si intende prevedere.
Una catena di Markov è raffigurata in Figura 5. Questa mostra due stati con le relative transizioni tra essi.

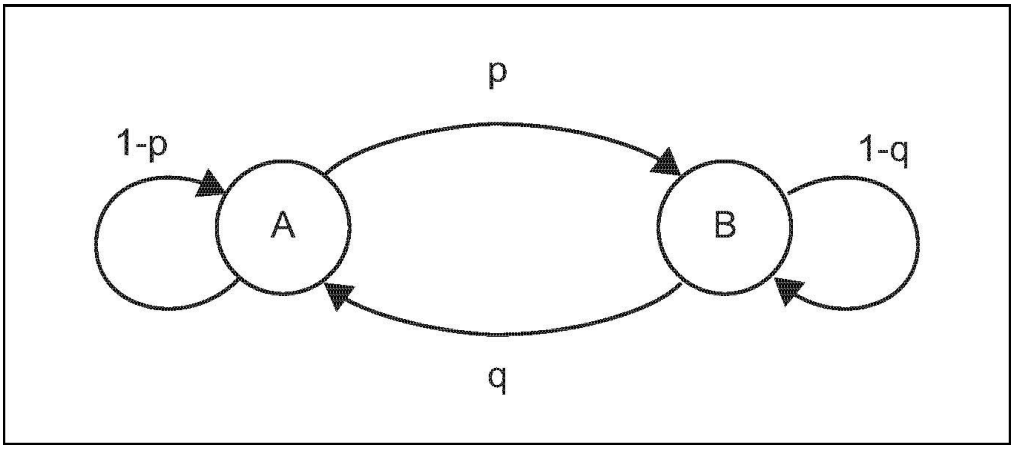

Figura 5: Catena di Markov con due stati.

Un modello di Markov nascosto è dunque un modello costituito da due processi stocastici diversi: un processo nascosto e un processo osservabile.
Il processo nascosto segue una catena di Markov e la relativa ipotesi; quello osservabile, dipende dal primo e descrive gli eventi osservabili.

L'Hidden Markov Model fu utilizzato per la prima volta, nella prima metà del '900, quando venne impiegato su Evgenij Onegin, romanzo scritto in versi dallo scrittore russo A. Puškin. Venne usato per studiare la probabilità che una lettera seguisse un'altra, e questo si è rivelato importante, perché ha mostrato come un processo apparentemente casuale (come la disposizione delle lettere in testi) poteva essere modellato con un modello matematico. Modelli probabilistici più complicati vennero analizzati tra gli anni '50 e '60, dove si iniziarono a studiare sistemi formati da stati non osservabili. Da qui si iniziò a parlare di "hidden state".

La vera nascita del modello di Markov nascosto si associa, però, a Leonard E. Baum et al. e ai loro studi all'istituto di Princeton nel New Jersey. Elaborarono anche alcuni degli algoritmi fondamentali per gli HMM: Baum-Welch e Viterbi. Una breve rappresentazione di un HMM è la Figura 6. Questa mostra come ogni stato, x_0, x_1, produce una sequenza di osservazioni, y_0, y_1.

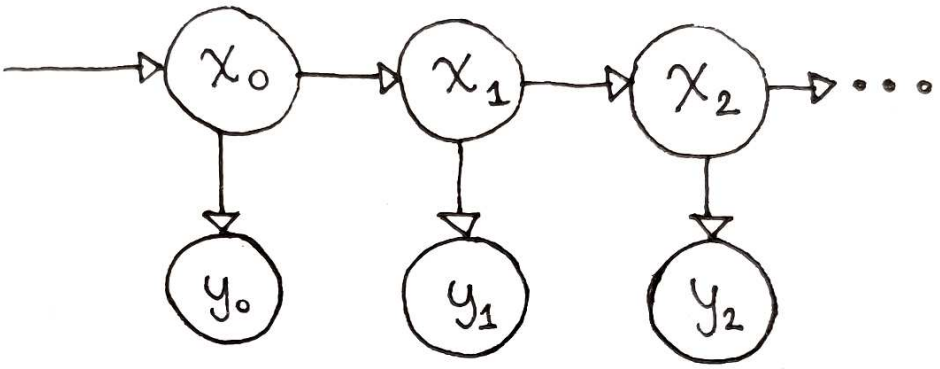

Figura 6: Hidden Markov Model - stati nascosti e osservabili.

Lo sviluppo di un modello di Markov nascosto richiede:

1. Una sequenza di stati nascosti: Sono le variabili latenti, quei fattori che influenzano le prestazioni del modello di NER, senza poter essere osservate direttamente.
2. Una sequenza di osservazioni: Sono le parole della frase, del testo, di cui si intende inferire l'entità.
3. Una probabilità di transizione: Rappresenta la probabilità di passare da uno stato nascosto ad un altro.
4. Una probabilità di emissione: Indica la probabilità di osservare una specifica parola da un particolare stato nascosto. Ad esempio, la probabilità che "Comacchio" appaia se lo stato è "GPE" (Entità geopolitica).
5. Una distribuzione iniziale di probabilità: È la probabilità che la catena di Markov inizi in ogni specifico stato nascosto.

Nel paragrafo successivo, ognuno di questi elementi verrà analizzato in dettaglio per capire come un modello di Markov nascosto risolve il problema del NER.

3.2 Implementazione del modello HMM per il NER

In questo paragrafo verrà studiato il modo che permette di risolvere il Named Entity Recognition mediante il modello di Markov nascosto [6]. Si studierà come gli HMM si adattano al NER, e come mediante l'algoritmo di Viterbi si riesca effettivamente a trovare la soluzione, a trovare le entità per ogni parola nella frase.

3.2.1 Definizione degli stati nascosti

In un modello di Markov nascosto gli stati nascosti sono rappresentati dalle entità, dalle categorie che devono essere inferite dal sistema. Nel named entity recognition gli stati nascosti rappresentano il legame che vi è tra una parola e un'entità, dove mediante un determinato formato di tagging, BIO o una sua variante, si riescono a cogliere le transizioni tra le varie entità.
Utilizzando il formato di tagging BIO, alcuni stati nascosti possono essere: B-Person, I-Person, B-Location, B-Facilities, O.

3.2.2 Le probabilità di Transizione e di Emissione

La probabilità di transizione rappresenta la probabilità che il modello si muova, che il modello cambi stato corrente. Vengono racchiuse in una matrice, $A = \{a_{ij}\}$, chiamata matrice di transizione. Ogni elemento della matrice, a_{ij}, rappresenta la probabilità che il modello passi dallo stato i allo stato j. Questa matrice viene detta "stocastica per riga", infatti la somma di tutti gli elementi di una riga è pari a 1.

In un modello di Markov nascosto si potrebbe avere ad esempio una probabilità $P(I - PER|B - PER)$ elevata, in quanto se la prima parola appartiene all'entità "Persona" è molto probabile che la parola successiva appartenga ancora alla stessa entità. Oppure si potrebbe avere una probabilità $P(O|I - LOC)$ anch'essa elevata, perché una volta terminata un'entità vi potrebbe essere una parola che non vi appartiene, vi potrebbe essere una parola come un verbo per transitare in un'altra entità.
In Figura 7 vi è un esempio di matrice di transizione, dove per definire gli stati, è stato utilizzato il formato BIO.

	B-PER	I-PER	B-LOC	I-LOC	O
B-PER	0.1	0.8	0.05	0	0.05
I-PER	0.05	0.85	0.05	0	0.05
B-LOC	0.05	0	0.1	0.8	0.05
I-LOC	0	0	0.05	0.9	0.05
O	0.1	0	0.1	0	0.8

Figura 7: Matrice di transizione.

Dalla Figura 7 si possono evincere alcune informazioni:

1. Se lo stato corrente è $B - PER$, è molto alta la probabilità di rimanere nella stessa entità andando nello stato $I - PER$.
2. Trovandosi in $I - LOC$, è nulla la probabilità di passare allo stato $I - PER$. Se non fosse così, significherebbe violare i principi del formato BIO, entrando in una nuova entità senza transitare per uno stato B o uno stato O.
3. Se lo stato attuale è O, è molto probabile rimanere nello stesso stato o transitare in un qualunque stato B.

La matrice di emissione, invece, permette di associare ad ogni stato nascosto ciascuna osservazione. Questa matrice mostra quanto è probabile osservare ciascun output in ciascuno stato. Un modello di Markov nascosto fondato su M osservazioni e N stati nascosti, avrà una matrice di emissione $B \in R^{NxM}$. Ciascun elemento della matrice, $b_j(k)$, indica la probabilità di osservare k trovandosi nello stato j. Una matrice di emissione è quella in Figura 8.

	Sting	è	cantante	inglese
PER	0.9	0.05	0.02	0.03
O	0.05	0.85	0.05	0.05
PROF	0.02	0.05	0.88	0.05
NAT	0.03	0.05	0.05	0.87

Figura 8: Matrice di Emissione.

Da questa si può inferire:

1. L'osservazione $Sting$ è molto probabile che sia associata allo stato PER, proprio perché è un nome di persona.
2. La parola e', voce del verbo essere, è molto probabile che non appartenga a nessuna entità.
3. Il termine $inglese$ è più verosimilmente una nazionalità rispetto alle altre opzioni possibili.

Entrambe le matrici, transizione ed emissione, operano insieme per inferire la sequenza di stati nascosti più probabile, iniziando dalla sequenza di osservazioni.

In Figura 9 viene bene illustrato come le probabilità di transizione a_{ij} dettano i passaggi da uno stato nascosto ad un altro, e come le probabilità di emissione b_{ij} la probabilità che uno stato generi una parola. Nei modelli di Markov nascosti, entrambe hanno un ruolo cruciale.

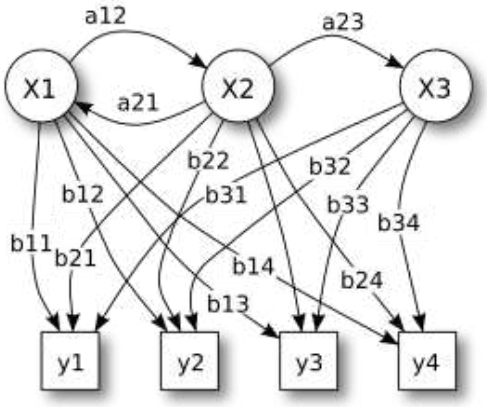

Figura 9: HMM - Probabilità iniziale, di transizione e di emissione.

L'algoritmo Forward - Backward [7], anche detto Baum - Welch, è quello che, data una sequenza di osservazioni, permette di calcolare queste due probabilità. Permette di apprendere la matrice di transizione e la matrice di emissione. È un algoritmo che appartiene alla categoria di apprendimento non supervisionato, ovvero quel learning in cui non vi sono dati di test. È un algoritmo iterativo (Figura 10) e prevede due diversi step: Expectation (E-step) e Maximization (M-step).

Given an input sequence and (S, K, Π, A, B)

1. Calculate forward probability:
   - Base case $\alpha_i(1) = \pi_i$
   - Recursive case: $\alpha_j(t+1) = \sum_i \alpha_i(t) a_{ij} b_{ijo_t}$
2. Calculate backward probability:
   - Base case: $\beta_i(T+1) = 1$
   - Recursive case: $\beta_i(t) = \sum_j \beta_j(t+1) a_{ij} b_{ijo_t}$
3. Calculate expected counts: $\xi_{ij}(t) = \frac{\alpha_i(t) a_{ij} b_{ijo_t} \beta_j(t+1)}{\sum_{m=1}^{N} \alpha_m(t) \beta_m(t)}$
4. Update the parameters:

$$a_{ij} = \frac{\sum_{t=1}^{T} \xi_{ij}(t)}{\sum_{j=1}^{N} \sum_{t=1}^{T} \xi_{ij}(t)} \qquad b_{ijk} = \frac{\sum_{t=1}^{T} \delta(o_t, w_k) \xi_{ij}(t)}{\sum_{t=1}^{T} \xi_{ij}(t)}$$

Figura 10: Step dell'algoritmo Forward - Backward

Inizia calcolando una stima delle due matrici A, B e successivamente alterna le due fasi:

A. E-Step: Tramite i parametri del modello, si determina la probabilità attesa di essere in un certo stato nascosto all'istante t e la probabilità attesa di una transizione tra stati nell'istante t. Questi due valori, vengono determinati mediante gli algoritmi Forward e Backward.
B. M-Step: Sulla base delle stime ottenute nella precedente fase, si va ad aggiornare i parametri del HMM, in modo tale da rendere massima la likelihood. Questo

aggiornamento, come visibile nella Figura 10, viene ricavato mediante una media pesata delle transizioni/ emissioni osservate.

Le due fasi si alternano ripetutamente, fino a quando non viene raggiunta la convergenza. Fino a quando la variazione nella likelihood tra due step successivi diventa minima. In Figura 11 è mostrato come avviene questo processo.

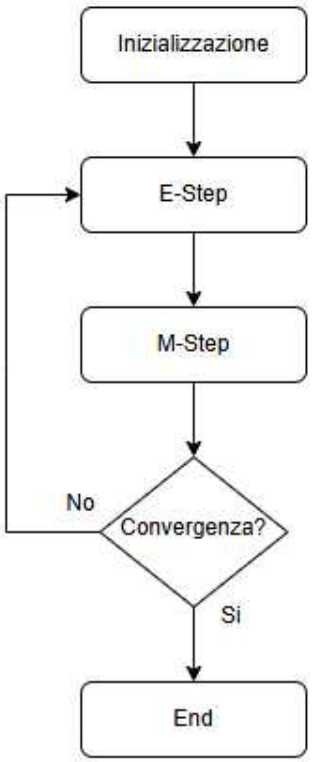

Figura 11: Diagramma di flusso dell'algoritmo Baum Welch.

3.2.3 Algoritmo di Viterbi

Viterbi è l'algoritmo ideato dall'ingegnere statunitense A. Viterbi per risolvere il problema della decodifica, ovvero data una sequenza di osservazioni formata da n parole, $(w_1, w_2, \ldots, w_n)$, determinare la sequenza di tag $(t_1, t_2, \ldots, t_n)$ più probabile.
L'obiettivo dell'algoritmo può essere dunque riformulato come segue:

$$t_{1 \ldots n} = arg\,max\, P(t_{1 \ldots n} \mid w_{1 \ldots n})$$

Il teorema di Bayes permette di riscrivere l'espressione come:

$$t_{1 \ldots n} = arg\,max\, \frac{P(w_{1 \ldots n} \mid t_{1 \ldots n}) * P(t_{1 \ldots n})}{P(w_{1 \ldots n})}$$

Consideriamo ora due ipotesi:

A. La probabilità di uno specifico tag dipende esclusivamente da quello precedente. Questa è l'ipotesi di Markov e la possiamo esprimere:

$$P(t_{1 \ldots n}) \approx \prod_{i=1}^{n} P(t_i \mid t_{i-1})$$

B. La probabilità di una parola dipende dal suo tag. Non da parole vicine e altri tag. Questa ipotesi è l'"indipendenza osservazione - tag" e ci permette di semplificare notevolmente la complessità del problema da risolvere. La possiamo indicare:

$$P(w_{1 \ldots n} \mid t_{1 \ldots n}) \approx \prod_{i=1}^{n} P(w_i \mid t_i)$$

Ritornando alla formulazione del problema, utilizzando le due ipotesi (a) e (b) otteniamo l'equazione per la sequenza di tag più probabile:

$$t_{1 \ldots n} = argmax \prod_{i=1}^{n} P(t_i \mid t_{i-1}) * P(w_i \mid t_i)$$

L'algoritmo di Viterbi si basa su quest'ultima formulazione.

È un algoritmo iterativo e fa uso di tutti e cinque gli strumenti richiesti da un Hidden Markov Model e definiti nel paragrafo precedente.

Esso prevede quattro passaggi:

1. Si calcola la probabilità iniziale per ciascuno stato.

$$P_i(1) = \pi_i * b_i(o_i)$$

   dove π_i è la probabilità iniziale dell' i-esimo stato e $b_i(o_i)$ la probabilità di emissione.
2. Per ciascun token si calcolano le probabilità di transizione a_{ij} e le probabilità di emissione $b_i(o_i)$.
3. Ad ogni passo temporale t, si calcola per ogni stato, la probabilità che questo sia attivato nel tempo t e che generi l'osservazione.

$$P_j(t) = max_i (P_i(t-1) * a_{ij}) * b_i(o_i)$$

   dove $P_i(t-1)$ è la probabilità di essere nell'i-esimo stato al tempo $t-1$. Si sceglie poi il massimo su tutti i possibili stati precedenti.
4. Si itera su ciascun token.

L'algoritmo richiede poi una certa struttura dati per ciascuno stato. Questa è richiesta per poter registrare il percorso migliore e salvare lo stato che massimizza la probabilità $P_j(t)$ del passo 3.

L'iterazione termina una volta eseguiti tutti i passaggi per ogni token. Alla fine viene effettuata la scelta dello stato con probabilità massima e grazie alla struttura dati si risale in backtracking alla sequenza di stati più probabile.

3.2.3.1 Esecuzione dell'algoritmo

In questo paragrafo viene mostrato come tramite Viterbi si riesca a etichettare una frase. Verrà mostrata l'esecuzione sulla frase "Cristiano Ronaldo è nato in Portogallo".

Per l'esecuzione dell'algoritmo di Viterbi è necessario conoscere a priori i fondamenti di un modello di Markov nascosto. Questi, come scritto nel paragrafo precedente sono: la matrice di transizione, quella di emissione e le probabilità iniziali. Tutte queste sono rappresentate di seguito:

	B-PER	I-PER	B-LOC	I-LOC	O
B-PER	0.1	0.8	0.05	0.01	0.04
I-PER	0.05	0.8	0.05	0.01	0.09
B-LOC	0.05	0.05	0.1	0.7	0.1
I-LOC	0.01	0.01	0.1	0.7	0.18
O	0.3	0.3	0.3	0.05	0.05

Tabella 1: Matrice di Transizione

	Cristiano	Ronaldo	è	nato	in	Portogallo
B-PER	0.9	0.8	0.01	0.01	0.01	0.05
I-PER	0.1	0.8	0.01	0.01	0.01	0.05
B-LOC	0.05	0.05	0.01	0.01	0.01	0.9
I-LOC	0.01	0.01	0.01	0.01	0.01	0.8
O	0.05	0.05	0.96	0.96	0.96	0.05

Tabella 2: Matrice di Emissione

$Stato$	π_i
$B-PER$	0.6
$I-PER$	0.0
$B-LOC$	0.2
$I-LOC$	0.0
O	0.2

Tabella 3: Probabilità iniziali

L'algoritmo costruisce una matrice, matrice di Viterbi. Ogni suo elemento è la probabilità massima di trovarsi in un certo stato, osservato un certo token della sequenza.
La prima colonna, è riempita con il prodotto tra le probabilità iniziali per le probabilità di emissione del primo token.
Vediamo come si esegue il calcolo:

$Stati$	π_i	$P(Cristiano\|s)$	$\pi_i \cdot P(Cristiano\|s)$
$B-PER$	0.6	0.9	0.54
$I-PER$	0.0	0.1	0.0
$B-LOC$	0.2	0.05	0.01
$I-LOC$	0.0	0.01	0.0
O	0.2	0.03	0.01

La matrice di Viterbi, dopo la prima iterazione, contiene solamente l'ultima colonna della tabella sopra riportata.

Ogni nuova iterazione consiste nell'aggiungere una colonna alla matrice. Ogni elemento del vettore colonna unisce le probabilità trovate al passo precedente con le probabilità di transizione e quelle di emissione del token corrente.
Tornando all'esempio, calcoliamo ora la probabilità massima di essere in B-PER alla parola "Ronaldo":

$$P(B\text{-}PER, 1) = \max \begin{bmatrix} 0.54 \cdot 0.1 \cdot 0.8 \\ 0 \cdot 0.05 \cdot 0.8 \\ 0.01 \cdot 0.05 \cdot 0.8 \\ 0 \cdot 0.01 \cdot 0.8 \\ 0.01 \cdot 0.3 \cdot 0.8 \end{bmatrix} = \max \begin{bmatrix} 0.0432 \\ 0 \\ 0.0004 \\ 0 \\ 0.0024 \end{bmatrix} = 0.0432$$

Ripetendo il calcolo per ogni stato, la matrice di Viterbi, dopo la seconda iterazione diventa:

$stato$	$P(s,0)$	$P(s,1)$
$B-PER$	0.54	0.0432
$I-PER$	0.0	0.3456
$B-LOC$	0.01	0.00135
$I-LOC$	0.0	0.00007
O	0.01	0.00108

Continuando l'esecuzione dell'algoritmo per tutti i token della frase e selezionando la probabilità massima in ogni colonna della matrice di Viterbi si ottiene la classificazione seguente:

Cristiano	*Ronaldo*	*è*	*nato*	*in*	*Portogallo*
B-PER	I-PER	O	O	O	B-LOC

Un ulteriore esecuzione pratica dell'algoritmo è disponibile al notebook Colab seguente. Qui è possibile visualizzare un implementazione dell'algoritmo e anche altri semplici esempi.
Link al notebook → Viterbi algorithm

3.3 Vantaggi e Limiti del HMM

Scoperto come il modello di Markov nascosto riesca a trovare la soluzione per il NER, in questo capitolo verranno esplorati alcuni dei principali pro [8] e anche dei contro rispetto l'HMM. Con questo paragrafo cercheremo quindi di capire il perché si sia passati a un'altra tipologia di soluzione, mantenendo questo, come l'approccio classico.

3.3.1 Quali sono i punti di forza?

Sono alcuni i vantaggi del modello di Markov nascosto. Sicuramente è un modello interpretabile facilmente. È un modello più semplice rispetto ad altri di machine learning o deep learning. Gli strumenti che utilizza, le probabilità di transizione, quella di emissione sono facilmente comprensibili, lo stesso vale per l'algoritmo alla base, Viterbi. Questo poi ha una complessità lineare rispetto alla sequenza di input. È quindi efficiente quando si debbono elaborare testi di lunghezza elevata. Un altro vantaggio è rappresentato dal fatto che questo modello sia indipendente dalla lingua. È un modello probabilistico, può quindi essere interpretato e utilizzato da diversi sviluppatori in contesti multilingue.

3.3.2 Quali sono i limiti?

I modelli di Markov nascosti presentano importanti limitazioni, e proprio per queste motivazioni sono state sviluppate nuove versioni e nuove soluzioni per risolvere il problema del Named Entity Recognition.
Gli HMM sono basati sull'assunzione di Markov, l'assenza di memoria. Questa è un'ipotesi molto forte. Permette di semplificare il problema, ma allo stesso potrebbe portare a trascurare informazioni, che, se considerate, potrebbero portare a un'etichettatura più appropriata. La scelta dell'etichetta migliore per una parola, molto spesso dipende anche dal contesto della parola nella frase, quindi, non considerarlo, potrebbe rivolgersi contro.
Come conseguenza dell'assunzione di Markov, vi è la problematica delle dipendenza a lungo raggio, ovvero quelle parole che possono essere influenzate da parole che sono posizionate molto prima o molto dopo nella frase. Gli HMM, non avendo una "memoria" rispetto agli stati precedenti, perdono queste informazioni.
La polisemia può essere un altro limite. Quelle parole che possono assumere vari significati. In questo caso, la mancata considerazione del contesto, può rivelarsi fatale.

Infine, vi sono le parole out of vocabulary (OOV). Sono quelle parole non contenute nel vocabolario del modello. Rappresentano un problema perché durante la decodifica si potrebbero incontrare parole con probabilità di emissione nulla. Possibili soluzioni a questo problema sono ad esempio l'uso, invece di probabilità nulle, di valori molto piccoli, oppure di introdurre un'etichetta apposita, UNK, per le parole OOV.

Il modello di Markov nascosto è, in conclusione, una soluzione probabilistica al NER. È un importante strumento in vari ambiti. Ma con la creazione delle reti neurali, in particolare delle reti neurali ricorrenti, è stata sviluppata una nuova strada per il sequence labeling. Grazie a particolari architetture di RNN, si è riusciti a superare i limiti propri del modello di Markov nascosto.

4. Approccio neurale: RNN per Sequence Labeling

Le reti neurali hanno rappresentato una sorta di “rivoluzione” rispetto ai modelli classici. Una rete neurale permette di creare modelli molto più complessi, ma che allo stesso tempo siano più robusti rispetto a una soluzione probabilistica. In questo quarto capitolo verrà approfondito come le reti neurali si possono usare per risolvere il Named Entity Recognition.

4.1 Introduzione alle reti neurali

Una rete neurale (NN) è un modello matematico costituito da neuroni artificiali. Una NN [9] vuole emulare il comportamento del cervello umano per arrivare a prestazioni cognitive che cercano di avvicinarsi a esso il più possibile. Lo sviluppo del primo neurone artificiale si deve a due americani, McCulloch e Pitts, nel 1943. Mentre per la prima rete neurale si è dovuto aspettare fino al 1975.

4.1.1 Struttura, Addestramento e il problema del gradiente

Una rete neurale (Fig. 12) è costituita da un layer di input, $(x_1, x_2, \ldots, x_n)$, che è lo strato iniziale, è quello che riceve i segnali, i dati dall’esterno.
Vi sono poi tanti pesi (Weights). Questi sono i parametri che stabiliscono le connessioni tra i vari neuroni. Sono rappresentati dagli archi diretti che collegano i nodi della rete. Il training delle NN prevede l’aggiornamento dei pesi tramite algoritmi come la Backpropagation. Una volta allenata, la rete è in grado di apprendere qualcosa e fare previsioni corrette.

Il ruolo cruciale è svolto dagli strati interni, hidden layers. Essi sono gli strati intermedi tra input e output. Trasformano i dati in ingresso, in informazioni che inviano agli strati successivi. NN con un singolo strato nascosto sono reti neurali “shallow”.
Al fine vi è lo strato di output, output layer $(y_1, y_2, \ldots, y_n)$. Questo rappresenta lo strato finale che deve produrre il risultato dei calcoli effettuati dai neuroni precedenti. Per problemi di classificazione si associa la softmax come funzione di attivazione.

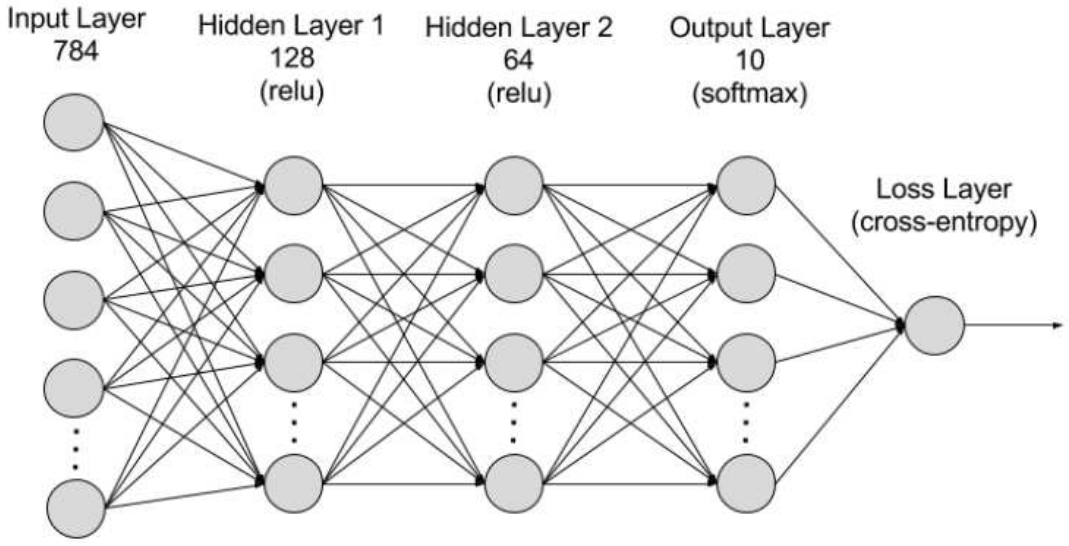

Figura 12: Deep neural network.

Una rete neurale viene addestrata tramite l'algoritmo Backpropagation [10]. Questa fase prevede l'aggiornamento della matrice dei pesi per cercare di minimizzare l'errore, che viene misurato con una funzione, chiamata loss. La loss function ci dice quanto è grande l'errore che il modello sta commettendo durante la previsione degli esempi di training. Durante l'addestramento è di cruciale importanza la scelta del momento in cui fermarsi. Questo non deve essere né troppo presto, ma neanche troppo tardi. Fermare l'addestramento presto significa andare in underfitting in cui si ha un modello che non riesce a cogliere la relazione tra input e output. Troppo tardi porta all'overfitting. Il modello si concentra troppo sui dati di training, perdendo la propria capacità di generalizzare su dati non visti.

La backpropagation percorre la rete dall'ultimo strato al primo, stabilendo come ogni parametro contribuisce all'errore totale. L'algoritmo fa uso del gradiente, che è un vettore contenente le derivate parziali. La derivata prima ci dice la direzione di maggior incremento. Quindi muovendosi nella direzione opposta del gradiente si può raggiungere un minimo, riducendo la loss (Gradient Descent).

Il gradient descent può portare a due problemi opposti (Fig. 13):

1. Vanishing gradient: Il gradiente viene calcolato con la chain rule, che è un metodo per calcolare le derivate di funzioni composte. In questo modo, il gradiente diventa un prodotto tra i gradienti. I primi strati della rete non producono grandi aggiornamenti dei pesi, infatti questi sono molto piccoli, quasi nulli. Quando la backpropagation arriva in questi strati iniziali si trova ad effettuare moltiplicazioni per valori prossimi allo zero. Questo porta il gradiente a scomparire.
2. Exploding gradient: Rappresenta il problema opposto rispetto al vanishing. Se i gradienti sono sempre più grandi, vi saranno moltiplicazioni per valori sempre maggiori, e questo porta l'algoritmo a divergere.

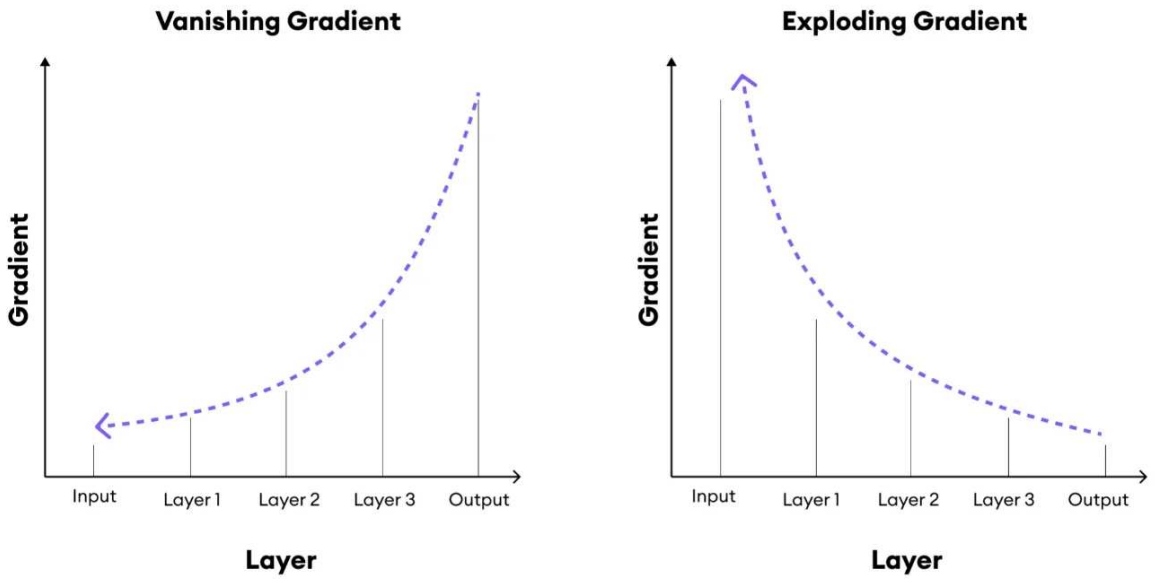

Figura 13: Gradient Descent - Vanishing e Exploding gradient.

4.2 Le RNN: Definizione e Architetture

Le prime reti neurali ricorrenti [11] vennero elaborate nella seconda metà degli anni ‘80. Qui i due lavori di maggior importanza furono la rete di Jordan del ‘86 e la rete di Elman del ‘90. Queste utilizzavano le RNN per studiare i processi mentali umani.

Una rete neurale ricorrente è descritta da una funzione del tipo:

$$h_t = f_W(h_{t-1})$$

dove h_t è il nuovo stato, f_W una funzione con certi parametri W e h_{t-1} il vecchio stato.
Da cui poi si ricava l’espressione finale:

$$h_t = f_W(h_{t-1}) = f_W(f_W(h_{t-2}))$$

Una RNN effettua una trasformazione dei dati in input in uno specifico output in ciascuna iterazione. Il vettore nascosto memorizza le coppie input- output precedenti in modo che ad ogni iterazione, la memoria viene modificata e viene aiutata l’elaborazione al passo successivo.

La Figura 14 metta a confronto una rete neurale classica con una RNN. Nella rete feedforward si può notare come i dati percorrano la rete sempre nella stessa direzione e che manca un meccanismo per ricordare le informazioni passate. Nella RNN, invece, certe uscite ritornano come input e grazie all’aggiornamento dello stato nascosto ad ogni iterazione si riescono a memorizzare informazioni passate.

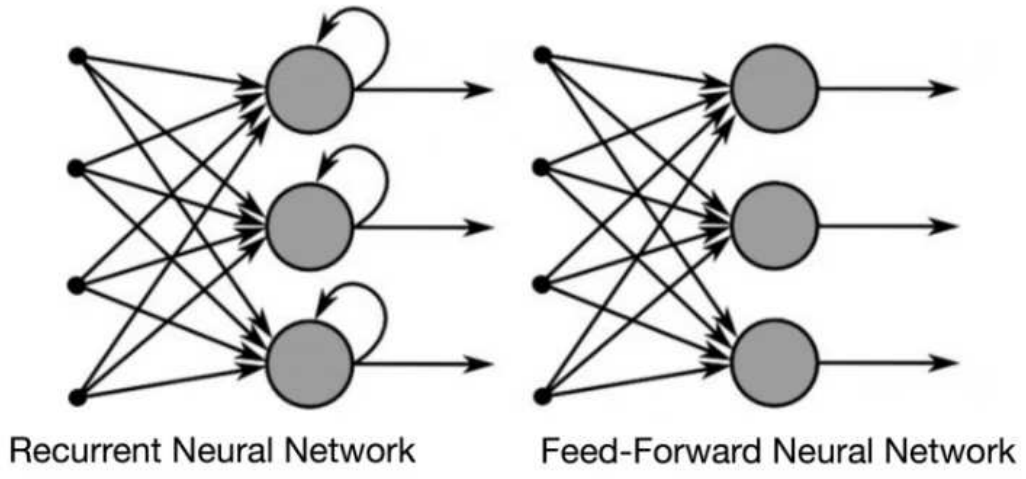

Figura 14: Confronto tra una rete neurale classica (Feed-Forward) e una RNN.

Le RNN [12] permettono di processare dati in forma sequenziale e sono in grado di riconoscere le relazioni tra i vari elementi. Ne esistono di tre tipologie e ognuna di queste è dedicata a risolvere un determinato problema.

1. One to Many: Uno specifico input produce diversi output.
2. Many to One: Rete dove, partendo da diversi input, si produce un singolo output.
3. Many to Many: Tipo di rete con cui si possono gestire sequenze di dati sia in ingresso che in uscita.

Nel Named Entity Recognition l'input è dato da una frase, un testo. Questa viene tokenizzata e così si ottiene una sequenza di tante parole. Per cui la tipologia one-to-many non è quella ideale, in quanto richiede un singolo input. L'obiettivo del NER è assegnare a ogni token della sequenza di input, un'etichetta. Da questo segue che la tipologia di RNN che più si adatta è many-to-many.

4.3 LSTM: Una soluzione neurale per il NER

Una rete Long Short Term Memory (LSTM) è una rete neurale ricorrente sviluppata per gestire le dipendenze a lungo raggio.

Nel Named Entity Recognition, abbiamo visto come una parola possa essere influenzata da quelle che all'interno della frase erano posizionate molto prima. Da questo punto di vista, una LSTM rappresenta la soluzione migliore. In questo sottocapitolo esamineremo come è strutturata questa tipologia di RNN e perché è una buona soluzione per il NER.

4.3.1 Struttura di una LSTM: Celle, Gate e Memoria

Una LSTM [13] è una rete neurale formata da tante unità come quella rappresentata in Figura 15. In ciascuna di queste unità il ruolo cruciale è svolto dal "cell state". Questo stato della cella rappresenta la memoria delle rete, attraversa l'intera sequenza e viene aggiornato solamente in due fasi. Queste due fasi applicano operazioni matematiche per decidere cosa deve essere ricordato e cosa può essere dimenticato.

A. Quello che può essere cancellato dalla memoria è gestito dalla porta "x". Questa è una moltiplicazione elemento per elemento (pointwise operation) applicata a vettori di ugual dimensione. Permette di annullare quei valori che non sono ritenuti più utili, liberando così parte di memoria che può essere usata per ricordare altre informazioni di importanza maggiore.
B. Le nuove informazioni da ricordare sono aggiunte dalla porta "+". Questa è una somma elemento per elemento, fonde i dati passati con quelli nuovi. Mantiene la continuità e l'evoluzione del contesto imparato lungo la sequenza.

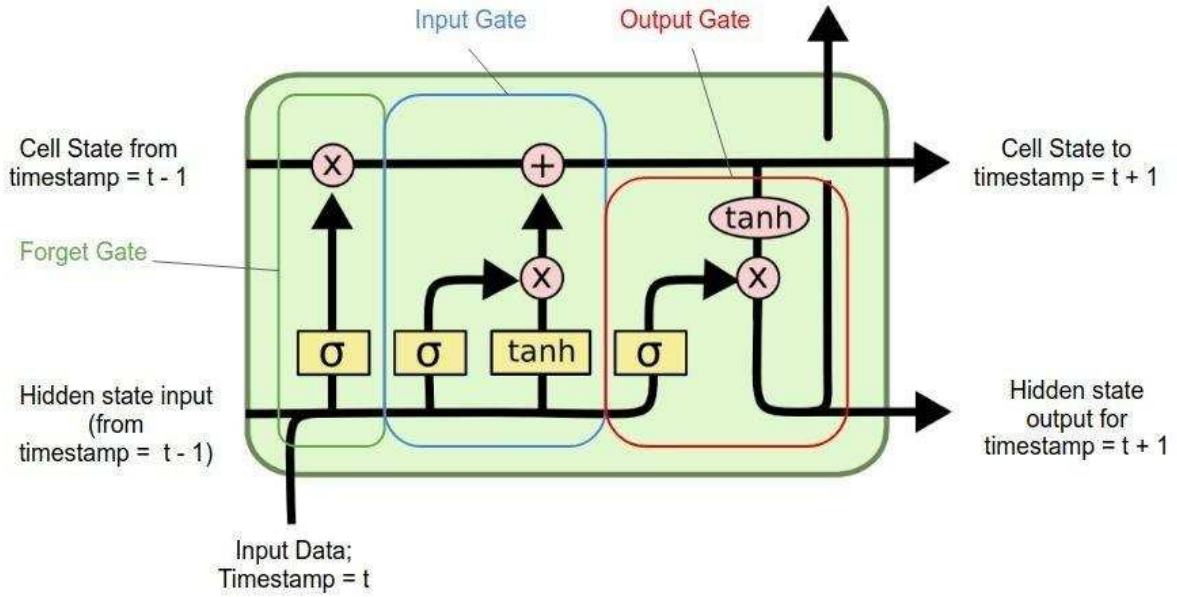

Figura 15: Singola unità di una LSTM

Oltre allo stato della cella, una LSTM è caratterizzata da tre diversi gate: Forget, Input, Output.

Il gate Forget decide quello che deve essere dimenticato, e che può essere cancellato dalla memoria. È un gate costituito da uno strato a cui è associata la funzione sigmoide σ. La funzione σ restituisce un valore appartenente all'intervallo $[0, 1]$, dove 0 significa che quell'input può essere dimenticato completamente, mentre 1 che quell'input è da ricordare integralmente. Matematicamente è definito dalla formula:

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$$

dove la matrice dei pesi è legata all'output precedente, h_{t-1}, e all'input attuale tramite una moltiplicazione matrice-vettore. Tale prodotto, produce un nuovo vettore che, addizionato al termine di bias b_f, viene mutato dalla sigmoide σ, per produrre il risultato del gate.

Il gate di Input decide quello che deve essere aggiunto alla memoria. È costituito da due diversi strati:

1. Input gate layer: Uno strato con funzione sigmoide che stabilisce i valori che devono essere aggiornati
$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$$
2. Gate gate layer: Uno strato con funzione tangente iperbolica che costruisce un vettore con i nuovi valori candidati.
$$\zeta_t = tanh(W_C * [h_{t-1}, x_t] + b_C)$$

Unendo forget e input gate, si ottiene che lo stato viene prima moltiplicato per il calcolo che è stato compiuto dal forget gate, poi gli viene aggiunta la parte calcolata dai due strati dell'input gate. Lo stato che si ottiene è:

$$C_t = f_t * C_{t-1} + i_t * \zeta_t$$

dove f_t indica se dimenticare o meno la parte "vecchia" e i_t se aggiungere o meno la parte "nuova".

Il gate di Output, infine, decide quello che deve essere inviato in output. L'output è basato sia sullo stato attuale, ma viene anche passato su due porte:

1. Una porta con tangente iperbolica tanh, che va a normalizzare i valori, portandoli in un intervallo $[-1, 1]$.
2. Una porta con funzione sigmoide che va a selezionare solamente le parti dell'output che devono essere conservate.
$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o)$$

L'output risulta:

$$h_t = o_t * tanh(C_t)$$

Anche le reti neurali LSTM, come le classiche reti neurali, vengono addestrate tramite backpropagation. Le LSTM, grazie alla loro architettura, riescono a ridurre di molto il problema del vanishing gradient. Infatti, durante la backpropagation il gradiente viene

moltiplicato elemento per elemento, invece che eseguire moltiplicazioni tra matrici. In questo modo le informazioni sono in grado di scorrere lungo le dipendenze.

Grazie a questa struttura, formata da gate e memoria interna, una rete LSTM riesce a gestire dipendenze a lungo raggio, garantendo un apprendimento buono anche in relazioni temporali complesse.

4.4 Costruzione di un modello neurale

Compresi i fondamenti delle reti neurali e delle reti ricorrenti, i paragrafi successivi mostrano come queste possono essere utilizzate per costruire un modello in grado di risolvere il problema del NER. In particolare, verrà analizzata la struttura del modello, le sue componenti e le attività principali necessarie.

4.4.1 Architettura

Il modello neurale più adatto a risolvere il Named Entity Recognition è una rete LSTM. Non una LSTM classica, ma una Bidirectional Long Short Term Memory. Questa rete analizza la sequenza in due direzioni: da sinistra a destra e viceversa. Si riesce così a catturare meglio il contesto di ciascuna parola e nel NER, l'etichetta da attribuire a ogni token può essere influenzata da parole sia precedenti ma anche da successive.

La rete utilizza, prima dello strato neurale, uno strato di embedding [14]. Questo strato è cruciale in quanto i parametri di input di una rete neurale debbono essere dei numeri, dei valori numerici. È necessario un metodo per tradurre sequenze di parole in numeri, cercando di non perdere informazioni importanti. Si potrebbe pensare di usare una rappresentazione One-hot-encoding, costruendo dei vettori che descrivono il corpus. Questi però non catturano le informazioni sulla distanza, sulla somiglianza tra le parole.

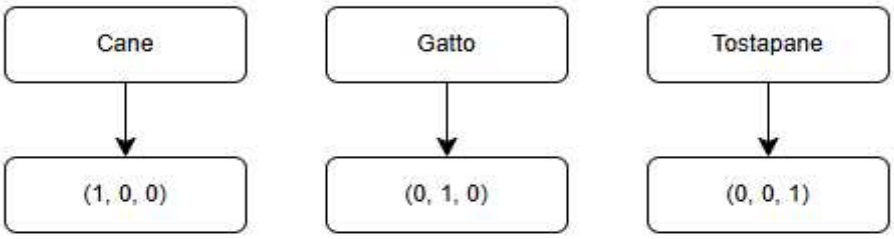

Figura 16: One-hot-encoding - distanza tra le parole

Come viene mostrato in Figura 16, usando una riproduzione di questo tipo, si può notare che i tre vettori sono ortogonali tra loro, e che la distanza tra ogni coppia è la medesima. Una rete neurale dovrebbe cogliere le differenze tra le varie parole, in questo caso, invece, la rete coglierebbe che la distanza tra parole come "tostapane" e "gatto" sarebbe la stessa.

Con gli Embeddings una parola viene espressa con un vettore m-dimensionale e le parole simili sono espresse con vettori vicini tra loro, come illustrato in Fig. 17. Questi possono essere d'aiuto anche come soluzione alla polisemia. Ad esempio si potrebbe avere che le parole "Java" e "Indonesia" possono essere vicine in una dimensione e "Java" e "Python" vicine in un'altra.

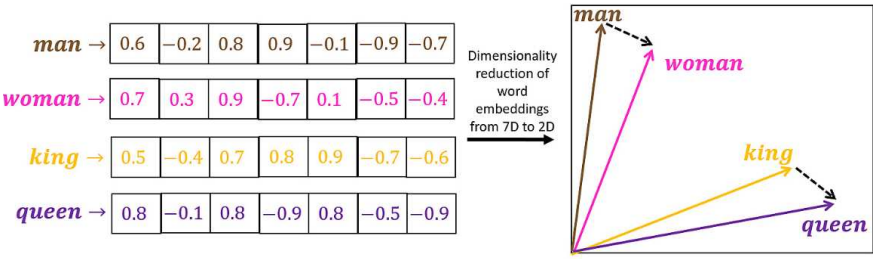

Figura 17: Embedding - Somiglianza semantica

Dopo lo strato Bi-LSTM, segue lo strato di output. Questo è lo strato che deve effettuare la classificazione per ciascun token. Può essere costruito con una funzione di attivazione softmax [15] che è l'estensione della funzione sigmoide applicata alla classificazione multiclasse. Softmax trasforma ogni vettore in valori che possono essere interpretati come probabilità. La softmax viene applicata a ogni token in modo indipendente, restituendo l'etichetta corretta.
Se si utilizzasse uno strato Flatten con alcuni strati densi con sempre softmax come funzione di attivazione, non si otterrebbe una soluzione efficace, in quanto il Flatten appiattisce la sequenza in un unico vettore, rendendo gli strati fully connected finale enormi e difficili da addestrare.

4.4.2 Dataset, Preprocessing e Training

La rete neurale definita nel paragrafo precedente viene utilizzata su un dataset. Questo dataset deve essere annotato, dato che questo problema appartiene alla categoria del learning supervisionato. Sono due i dataset più adoperati: CoNLL-2003 o OntoNotes 5.0.

CoNLL-2003 [16] è un dataset contenente dati in due diverse lingue: inglese e tedesco. È suddiviso inoltre in training set e validation set. I dati in lingua inglese provengono da articoli "Reuters" risalenti al periodo compreso tra agosto '96 e dicembre dello stesso anno. Mentre i dati in lingua tedesca risalgono al rotocalco "Frankfurter Rundschau".
OntoNotes 5.0 [17] è invece un dataset molto più ampio che contiene dati provenienti da blog, trasmissioni e talk show. Dati che sono disponibili in tre lingue: inglese, arabo e cinese.

I dati, prima di diventare l'input della rete neurale, soggetti a operazioni di preprocessing. Questa attività [18] è necessaria per trasformare i dati in un formato coerente con quello richiesto dalla rete. Le frasi vengono prima tokenizzate, viene quindi creata una sequenza formata dalle unità minime di significato. Successivamente si procede a una normalizzazione e standardizzazione. Queste tecniche sono importanti per rendere migliore la capacità di generalizzare e per avere prestazioni migliori. Con queste operazioni si riesce a costruire un insieme di dati che può essere fornito come input.

Elaborati i dati, definito il modello, si addestra la rete. L'apprendimento avviene tramite Backpropagation come in una classica rete neurale. Se viene utilizzato lo strato finale softmax, allora come loss function si utilizzerà la categorical cross entropy. Durante la fase di training di una rete è quasi obbligatoria la scelta di un ottimizzatore, ovvero scegliere un metodo che produca migliori prestazioni, aggiornando i pesi. Quelli usati più frequentemente sono Adam o RMSProp, che riescono ad adattare il learning rate a seconda dell'iterazione.
È inoltre classica la scelta di una tecnica di regolarizzazione che permette di ridurre l'errore di test a scapito di un possibile aumento di quello di training. Ad esempio, si può scegliere tra dropout o early stopping, dove la prima consiste nel "spegnere" a caso certe unità in modo che il modello non si affidi a specifici pattern, mentre il secondo, prevede l'arresto anticipato del training quando non si verificano nuovi miglioramenti per un certo numero di epoche consecutive. Per valutare le performance del training si utilizzano delle metriche. Quelle più usate sono l'accuracy, buona solo quando il dataset è bilanciato, o F-Measure, che è una media armonica di precision (quante entità sono corrette) e recall (quante delle entità totali sono state correttamente riconosciute).

La struttura completa del modello è mostrata in Figura 18. Qui sono state raccolte tutte le fasi necessarie per passare da una frase scritta in linguaggio naturale all'attribuzione delle etichette per ogni parola.

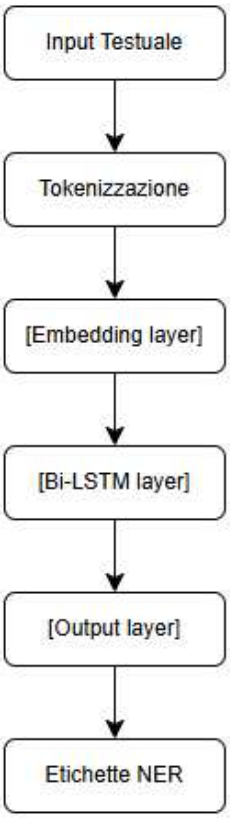

Figura 18: Pipeline di un modello neurale per il NER

In letteratura, sono diversi gli approcci presentati per il Named Entity Recognition, uno è quello proposto da Lample et al. [19] nel giugno del 2016. La loro soluzione prevedeva una rete neurale Bi-LSTM con un livello finale costituito da CRF (Conditional Random Field), un modello basato su probabilità per la previsione di etichette.

4.5 Soluzioni neurali a confronto

Di reti neurali ne sono state sviluppate di vario tipo, più o meno adatte al Named Entity Recognition. In questo capitolo verranno paragonate diverse architetture utilizzando, come illustrato precedentemente, metriche quali F-Measure e Accuracy. Con questi parametri possono essere valutate le prestazioni di un qualunque modello di Machine o Deep Learning, compresa anche una rete neurale.

Una rete ricorrente classica (Vanilla RNN) è una rete sviluppata per poter gestire input di tipo sequenziale. Sarebbe un buon metodo per il Named Entity Recognition, in quanto la memoria viene aggiornata step-by-step, ma risentendo del vanishing gradient l'apprendimento diventa difficile sulle lunghe distanze. Qui il punteggio sia in Accuracy che F-Measure è basso, è poco inferiore al 70%.

Una Long Short Term Memory e in particolare una Bi-LSTM rappresentano un notevole miglioramento. Il fatto di leggere contesto in entrambe le direzioni porta a valori che si attestano intorno all'90% su entrambe le metriche.

Una variante è la LSTM con Spyholes dove ai vari gate sono aggiunte connessioni dirette alla cella di memoria, così i gate possono "curiosare" dentro la memoria prima di decidere quello da eliminare e mantenere. Sono una variante che introduce complessità alla rete, e questa non è pienamente compensata dai risultati che possono essere ottenuti.

Un'altra alternativa sono le Clockwork RNN. È una rete che seziona le unità ricorrenti in moduli, e questi moduli aggiornano il proprio stato a frequenze differenti. È una rete in grado di gestire fenomeni con diverse velocità. Utilizzare questa tipologia per risolvere il NER non porta a un cambiamento vantaggioso rispetto a una Bi-LSTM. Si possono ottenere valori intorno all'85 - 88%.

Questi risultati[1], racchiusi nella tabella sottostante, sono tutti ricavati sul dataset CoNLL-2003.

Modello	Vantaggio principale	Accuracy	F-Measure
Vanilla RNN	Sequenze semplici	65-70%	65-68%
LSTM	Gestione delle dipendenze lunghe	80-83%	80-82%
Bi-LSTM	Stato dell'arte nel NER	89-91%	90%
LSTM with Spyholes	Migliore controllo dell'informazione	80-82%	81-83%
Clockwork RNN	Cattura dipendenze a varie scale temporali	n/a	85-87%

Figura 19: Reti neurali a confronto.

Si può notare dunque che tra le opzioni, quella più adatta, sia come complessità che come prestazioni è la Bi-LSTM.

[1] I risultati su Vanilla RNN, LSTM e Bi-LSTM provengono da Lample et al. (2016). LSTM with Spyholes da Gers & Schmidhuber (2000). Infine gli esiti su Clockwork RNN da Koutnik et al. (2014).

Conclusioni e prospettive future

Nel seguente approfondimento sono state approfondite alcune delle principali tecniche per esaminare il Named Entity Recognition.
L'approfondimento ha proposto un "confronto" tra quello che viene considerato l'approccio classico mediante un modello probabilistico come il modello di Markov nascosto (HMM) e l'approccio neurale mediante le reti neurali ricorrenti (RNN).

Il modello di Markov nascosto è un modello semplice, interpretabile e in generale adatto a risolvere problemi poco articolati. Le motivazioni sono dovute proprio alla sua costituzione. L'ipotesi di Markov limita fortemente la propria capacità di plasmare il contesto.

Le RNN, ed a maggior ragione le Bidirectional-LSTM garantiscono una capacità molto più elevata grazie alla loro struttura formata da gate. Una rete neurale è un modello che è meno interpretabile, ma questo viene compensato dai validi risultati che si possono ottenere.

La tabella sottostante racchiude in modo schematico questi vantaggi e svantaggi appena elencati.

Criterio	HMM	Bi-LSTM
Apprendimento	Non supervisionato (Baum-Welch)	Supervisionato (Backprop)
Memoria	Limitata (1 passo)	Estesa
Interpretabilità	Alta	Bassa
Complessità	Bassa	Alta
Accuratezza nel NER	65-70%	90%
Robustezza al rumore	Bassa	Alta

Figura 20: HMM - Bi-LSTM: Confronto conclusivo

Le reti Bi-LSTM si dimostrano dunque la scelta migliore su quasi tutti i fronti analizzati. Bisogna anche sottolineare il fatto che, per il Named Entity Recognition negli ultimi anni, si stanno facendo importanti passi avanti. Lo sviluppo di architetture come i Transformer, che fanno uso di altri meccanismi, permettono di raggiungere prestazioni ancora migliori, a fronte di risorse computazionali elevate.

In conclusione, conoscere le basi probabilistiche può essere comunque importante sia per riuscire a comprendere le soluzioni più complesse, che per scegliere il modello migliore a seconda del contesto applicativo.

Bibliografia

Fonti testuali

[1] IBM - What is named entity recognition? - https://www.ibm.com/think/topics/named-entity-recognition - Accesso: 9/5/2025

[2] Shaip - Cos'è il riconoscimento delle entità nominate (NER) - Esempio, casi d'uso, vantaggi e sfide - https://it.shaip.com/blog/named-entity-recognition-and-its-types/ - Accesso: 9/5/2025

[3] Wikipedia - Sequence Labeling - https://en.wikipedia.org/wiki/Sequence_labeling - Accesso: 9/5/2025

[4] Medium - M. Maurya - Name Entity Recognition and various tagging schemes - https://medium.com/@muskaan.maurya06/name-entity-recognition-and-various-tagging-schemes-533f2ac99f52 - Accesso: 9/5/2025

[5] Stanford University - Hidden Markov Model - https://web.stanford.edu/~jurafsky/slp3/A.pdf - Accesso: 10/5/2025

[6] HumAI - Il Pos Tagging con il modello HMM (Hidden Markov Model) - https://www.humai.it/semplifichiamo/il-pos-tagging-con-il-modello-hmm-hidden-markov-model/ - Accesso: 10/5/2025

[7] Wikipedia - Algoritmo di Baum-Welch - https://it.wikipedia.org/wiki/Algoritmo_di_Baum-Welch - Accesso:14/06/2025

[8] Medium - A. Verma - Unveiling the Hidden Markov Model in NLP - https://ai.plainenglish.io/unveiling-the-hidden-markov-model-in-nlp-bc469e3c3fce - Accesso: 10/5/2025

[9] AWS - Cos'è una rete neurale? - https://aws.amazon.com/it/what-is/neural-network/ - Accesso: 10/5/2025

[10] M. Speciale - Diario di un Analista - Backpropagation: come apprendono le reti neurali - https://www.diariodiunanalista.it/posts/backpropagation-come-apprendono-le-reti-neurali/ - Accesso: 10/5/2025

[11] Wikipedia - Recurrent neural network - https://en.wikipedia.org/wiki/Recurrent_neural_network - Accesso: 11/5/2025

[12] Servicenow - Che cos'è una rete neurale ricorrente (RNN)? - https://www.servicenow.com/it/ai/what-are-recurring-neural-network.html - Accesso: 11/5/2025

[13] Analytics Vidhya - An Overview on Long Short Term Memory (LSTM) - https://www.analyticsvidhya.com/blog/2022/03/an-overview-on-long-short-term-memory-lstm/ - Accesso: 20/5/2025

[14] Responsa - Embeddings: nella mente di una rete neurale - https://responsa.ai/embeddings-nella-mente-di-una-rete-neurale/ - Accesso: 15/06/2025

[15] Stack Overflow - Keras TimeDistributed Dense with softmax is not normalised per time step - https://stackoverflow.com/questions/54533820/keras-timedistributed-dense-with-softmax-is-not-normalised-per-time-step?utm_source=chatgpt.com - Accesso: 15/06/2025

[16] CoNLL 2003 - https://paperswithcode.com/dataset/conll-2003 - Accesso: 15/06/2025

[17] OntoNotes 5.0 - https://paperswithcode.com/dataset/ontonotes-5-0 - Accesso: 15/06/2025

[18] Bnova - Data Preprocessing: che cos'è, a cosa serve e come si fa - https://www.bnova.it/data-science/data-preprocessing/ - Accesso: 15/06/2025

[19] ACL Anthology - Neural Architectures for Named Entity Recognition - https://aclanthology.org/N16-1030/ - Accesso: 15/06/2025

Fonti iconografiche

[A] Figura copertina - John Snow Labs - An Overview of Named Entity Recognition (NER) in NLP with Examples - https://www.johnsnowlabs.com/an-overview-of-named-entity-recognition-in-natural-language-processing/ - Accesso: 9/5/2025

[B] Figura 2 - Techslang - What Is Named Entity Recognition (NER)? - https://www.techslang.com/definition/what-is-named-entity-recognition-ner/ - Accesso: 21/5/2025

[C] Figura 5 - Treccani - Markov, catena di Enciclopedia della Matematica (2013) - https://www.treccani.it/enciclopedia/catena-di-markov_%28Enciclopedia-della-Matematica%29/ - Accesso: 10/5/2025

[D] Figura 6 - Avik Das - Dynamic programming for machine learning: Hidden Markov Models - https://avikdas.com/2019/06/24/dynamic-programming-for-machine-learning-hidden-markov-models.html - Accesso: 10/5/2025

[E] Figura 9 - Wikipedia - Hidden Markov Model - https://en.wikipedia.org/wiki/Hidden_Markov_model - Accesso: 10/5/2025

[F] Figura 10 - SlideServe - Forward - Backward algorithm - https://www.slideserve.com/jana/forward-backward-algorithm - Accesso: 20/05/2025

[G] Figura 12 - AWS - Cos'è una Rete Neurale? - https://aws.amazon.com/it/what-is/neural-network/ - Accesso: 10/5/2025

[H] Figura 13 - AIML.com - What is the vanishing and exploding gradient problem, and how are they typically addressed? - https://aiml.com/what-do-you-mean-by-vanishing-and-exploding-gradient-problem-and-how-are-they-typically-addressed/ - Accesso: 11/5/2025

[I] Figura 14 - Intelligenza Artificiale Italia - Cosa sono le reti neurali ricorrenti RNN? Un'introduzione alle reti neurali ricorrenti - https://www.intelligenzaartificialeitalia.net/post/cosa-sono-le-reti-neurali-ricorrenti-rnn-un-introduzione-alle-reti-neurali-ricorrenti - Accesso: 11/5/2025

[J] Figura 15 - LinkedIn - LSTM Networks - https://www.linkedin.com/pulse/lstm-networks-abdullah-al-rahman - Accesso: 20/05/2025

[K] Figura 17 - Airbyte - What are Word & Sentence Embedding? 5 Applications - https://airbyte.com/data-engineering-resources/sentence-word-embeddings - Accesso: 15/06/2025