Riconoscimento automatico della Prominenza in Italiano: Reti Neurali Ricorsive

Maxim Gaina

Università di Bologna*
maxim.gaina@studio.unibo.it

28 aprile 2017

Sommario

Nel parlato, la prominenza è un fenomeno percettivo che può influire pesantemente sulla semantica delle frasi pronunciate, per questo guadagna un ruolo fondamentale nella comunicazione umana. Sono stati svolti in passato diversi sperimenti per rilevare le sillabe prominenti all'interno di una frase, che prevedevano l'utilizzo di svariati strumenti offerti dall'ambito dell'Apprendimento Automatico e dai modelli Rule-Based. L'obiettivo di questo lavoro di progetto è impiegare, per la prima volta nella lingua italiana, le reti neurali ricorsive; valutare successivamente i benefici e infine confrontarli con i metodi precedentemente usati.

Indice

*Progetto per il corso di Elaborazione del Linguaggio Naturale, A.A. 2016/2017, prof. Fabio Tamburini.

Introduzione

Ogni lingua possiede un suo insieme di tratti soprasegmentali, che caratterizzano la relativa sequenza lineare del parlato, e i vari rapporti tra foni che compongono tale sequenza. Tali informazioni prosodiche sono spesso cruciali per disambiguare espressioni sintattiche. Per capire, in breve, si prenda l'esempio riportato in [1]:

a. *È stato Luigi?*
b. *È stato Luigi.*

In italiano le frasi interrogative sono un esempio di come il profilo intonativo del parlante sia fondamentale, perché sintatticamente non sono marcate in alcun modo. Nel primo caso si ha un andamento finale ascendente, mentre nella frase dichiarativa tipicamente si tratta di un profilo finale discendente. In altri casi invece, il parlante accentua e/o allunga determinate parole, per concentrare l'attenzione dell'ascoltatore proprio in quel punto della sequenza. Si tratta ancora di informazioni difficilmente trasmissibili o per niente trasmissi-

bili per vie sintattiche. I fenomeni prosidici sono quindi l'intonazione, il ritmo, la velocità di elocuzione, le pause e così via. L'oggetto di studio in questo lavoro è la **prominenza**, per la quale, più tardi, si cercherà di dare una definizione più esatta. Lo scopo ultimo, è quello di impiegare un particolare tipo di *Reti Neurali Ricorsive* (RNN), cioè le *Long-Short Term Memory* per l'identificazione automatica delle sillabe prominenti. Ancora meglio, si può dire che questo sia un problema di classificazione delle sillabe, prominenti e non, dove le caratteristiche di ogni sillaba dipendono dall'intero contesto in cui si trovano.

Verrà qui esposto il modo in cui è stato affrontato il problema d'esame, preceduto però dalle informazioni necessarie allo studente per capire il problema trattato e il suo contesto, cioè dove si colloca all'interno del *Natural Language Processing*. La relazione avrà quindi la seguente struttura:

- nella prima sezione si cercherà di definire meglio il dominio del problema, restringendolo;
- nella seconda sezione verranno spiegati brevemente i metodi di risoluzione del problema, in ambito NLP;
- la terza sezione descriverà brevemente un particolare lavoro di sperimentazione già eseguito, e grazie al quale si ha a disposizione un corpus;
- la quarta sezione, sarà quella che approfondisce il nuovo strumento che si vuole usare per la costruzione del modello, cioè le *Reti Neurali Ricorsive*;
- verrà poi spiegato nello specifico il metodo di risoluzione e visualizzati i risultati ottenuti nella quinta sezione.

I. Il Fenomeno della Prominenza

È stato detto che si vuole identificare automaticamente la prominenza, ma cosa significa esattamente? Non è proprio immediato dare una definizione, dal momento che gli studiosi addentratisi nel problema, arrivarono spesso a conclusioni divergenti, proponendo teorie e metodi diversi. Esaminando qualche rasségna in merito ([1, Capitolo 1] e [3]) è possibile vederlo. Talvolta i ricercatori vanno letteralmente in conflitto ridefinendo lo stesso termine, come per esempio lo *stress*. Riprendendo però quanto detto in [3], si può così definire la prominenza:

Definizione 1 (Prominenza)**.** *La prominenza è un fenomeno percettivo, continuo nella sua natura, che enfatizza alcune unità linguistiche e segmentali rispettivamente al contesto che li circonda, e viene supportata da una complessa interazione fra parametri prosodici e fonetico-acustici.*

Da notare come la definizione fa riferimento a unità linguistiche segmentali all'interno dell'enunciato e, che queste siano dipendenti da altre unità linguistiche, che formano il contesto in cui si trovano. Questo sarà decisivo nel definire poi il metodo di risoluzione del problema. I parametri a cui si fa riferimento sono anche essi sempre stati oggetto di discussione. Uno degli obiettivi più importanti è sempre stato trovare un fenomeno linguistico/prosodico misurabile per supportare la percezione della prominenza. Fra tutti, e come suggerito in [3], si riporta come esempio uno studio che individua due impportanti attori al supporto della prominenza: il primo è il *pitch accent* e riguarda il profilo della frequenza fondamentale; il secondo è il *force accent*, più connesso a fenomeni acustici come intensità, durata del segmento e probabilmente altro. Il grado di prominenza dell'i-esimo segmento si può vedere come la somma di questi due parametri.

Ci si potrebbe ora chiedere se la prominenza di un segmento si può stabilire con un *sì/no*, su una scala ordinale oppure su una scala continua. La prominenza per sua natura è un fenomeno continuo, ma la sua discretizzazione può comunque essere attuata, dipende dall'approccio che si adotta per risolvere il problema.

Un'altra domanda che sorge, è se lo studio della prominenza esige che vengano fatte delle distinzioni interlinguistiche. La risposta è sì, in quanto si può rilevare la presenza delle così dette *tone languages*, in cui il profilo del pitch

è correlato al significato stesso della parola pronunciata (il tailandese). Una versione più rilassata di questo principio è, per esempio, il giapponese, dove questo ancoraggio fra prominenza e semantica è molto più limitato, e le sillabe prominenti non portano con sé un allungamento o una profondità maggiore, ma semplicemente un profilo di pitch alto. Infine troviamo le *stress languages*, nelle quali il tono non incide in alcun modo sui significati lessicali delle parole (inglese), ma le sillabe vengono *stressate* in funzione comunicativa. Questo discorso è stato approfodito ulteriormente in alcuni studi, che classificano le lingue in funzione di due dimensioni: prominenza e pattern ritmico. La prima dimensione è importante da considerare, mentre la seconda interessa poco in questa sede.

II. Riconoscimento Automatico della Prominenza

Una volta definito e indagato il fenomeno della prominenza, è ora di chiedersi in che modo possa essere affrontato il problema del suo riconoscimento automatico. Si possono impiegare modelli computazionali che seguono due paradigmi diversi:

1. *Rule Based*, consiste nel fornire regole formali per la risoluzione di un dato problema;

2. *Machine Learning*, viene retto un modello computazionale a partire da dati già esistenti, in questo caso da corpus annotati da esperti.

Riprendendo il comodo schema proposto in [3], si può dire che per quanto riguarda i sistemi rule-based, i principali vantaggi consistono nel permettere a chi si occupa di linguistica di controllare pienamente il comportamento degli algoritmi; di creare modelli che operano su diverse lingue; non richiedono grossi corpora annotati, e permettono di epsrimere il modello in termini linguistici. Solitamente però producono sistemi meno accurati. Si escludono tuttavia i sistemi rule-based in questo contesto, e ci si concentra sui metodi di apprendimento automatico. Questi ultimi permettono di avere sistemi altamente performanti; permettono una più veloce classificazione e apprendono a partire dai dati. L'ultimo aspetto elencato è anche un punto dolente in `NLP`, sono infatti necessari grossi corpora annotati che non sempre si hanno a disposizione. La creazione di corpora può richiedere enormi sforzi umani ed economici, in più gli annotatori umani non sono mai al 100% concordi sulla parte prominente delle frasi. Si noti che in ambito *Machine Learning* (`ML`) esistono metodi di apprendimento supervisionati e metodi apprendimento non supervisionati, e la necessità di avere queste risorse annotate è presente nei sistemi supervisionati. In [1, Sezione 4.2] è stato proposto un metodo non supervisionato, che, data la segmentazione dell'enunciato, fa uso di una funzione di prominenza per fissarla su una scala continua. L'obiettivo di sistemi del genere è quello di liberare dalla necessità di avere larghi corpus annotati. In questo lavoro si cercherà di costruire tuttavia, un sistema supervisionato con diverse fasi di apprendimento.

Per quanto riguarda la segmentazione degli enunciati, il problema è simile. Se il corpus è creato da risorse umane si possono usare le segmentazioni eseguite a mano. Negli ambienti più realistici però, queste informazioni non si hanno a disposizione. Anche in merito sono stati fatti degli studi per identificare automaticamente delle pseudo sillabe e i relativi nuclei.

i. Metriche di valutazione

Si intende in questa relazione riportare i risultati di sperimentazioni già fatte e confrontarli con il lavoro qui svolto, per questo è necessario usare delle metriche ben precise. Tali metriche sono tipiche dell'ambito `ML`, e l'importanza di ognuna di loro varia a seconda del tipo di problema e quindi dei modelli usati. Verranno ora viste quelle fondamentali per affrontare il problema posto.

Dato un insieme di sillabe, siano i seguenti totali calcolati:

- a: corrette previsioni di non prominenza;
- b: scorrette previsioni di prominenza;
- c: scorrette previsioni di non prominenza;
- d: corrette previsioni della prominenza.

In altre parole sarebbero, rispettivamente, le sillabe scartate correttamente; i falsi allarmi; le mancate individuazioni e le corrette individuazioni della prominenza. A partire da questi parametri si può già definire l'**accuratezza** AC, che esprime la percentuale di corrette classificazioni rispetto al totale numero delle sillabe:

$$AC = (a + d)/(a + b + c + d) \tag{1}$$

La metrica **recupero** R invece, abbondantemente usata anche nell'ambito dell'*Information Retrieval* (per esprimere la percentuale di documenti utili trovati sul totale di quelli rilevanti), rappresenta qui la percentuale di sillabe prominenti correttamente identificate, ovvero:

$$R = d/(c + d) \tag{2}$$

La **precisione** P è la percentuale di corrette classificazioni di sillabe prominenti:

$$P = d/(b + d) \tag{3}$$

Quando gli elementi da classificare sono distribuiti in modo sbilanciato, la metrica AC può non essere il miglior indicatore delle prestazioni del modello. Si vedrà infatti, anche se è intuibile, che le sillabe prominenti sono una minoranza. Siamo anche di fronte a una situazione in cui, per esempio, il richiamo R come metrica non è strettamente più importante della precisione. Per valutare le prestazioni di un sistema, si ricorre quindi a una metrica combinata che tiene conto di entrambe, che può essere l'**F-Score** così definito:

$$F1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{4}$$

Viene chiamata $F1$ perché si tratta del caso specifico in cui nella formula generalizzata dell'*F-score* β viene impostato a 1, così che precisione e richiamo abbiano lo stesso peso.

III. Il Corpus Annotato

Per raggiungere l'obiettivo verrà usato lo stesso corpus dell'eperiménto [2], che ora si andrà ad analizzare per avere un'adeguata comprensione.

i. L'esperimento orignario

Esiste un'altra famiglia di sistemi `ML`, le *Probabilistic Graphical Models* (`PGM`). Esse sono in grado di elaborare sequenze di dati in input facendo previsioni su quelle in output, considerando sia la sequenza di input attuale che quella di output precedente. Data quindi una generica sequenza di vettori $\{v_1^x, ..., v_n^x\}$ in input, contenenti le features, e data la sequenza $\{y_1, ..., y_n\}$ in output, i modelli di tipo `PGM` etichettano ogni y_i nel modo più probabilmente opprtuno. Le `PGM` però sono un insieme di convinzioni per strutturare i modelli, e più tipi di `PGM` sono state usate in [2] per il riconoscimento automatico della prominenza.

Il corpus è stato costruito a partire da conversazioni di persone italiane provenienti dall'area di Pisa, sia femmine che maschi. Il corpus contiene un totale di 120 espressioni segmentate, con una lunghezza media di 18 sillabe che vanno da un minimo di 9 a un massimo di 35. Ai partecipanti fu chiesto poi di individuare quali fosse secondo loro le prominenze negli enunciati, presentando loro le segmentazioni. L'intero procedimento, in realtà più complesso, è descritto nel documento originale. Le sillabe sono state poi raggruppate in base alla percentuale di convergenza di etichettature da parte di tutti. Questi significa che, dato che ogni sillaba è stata valutata da 10 persone, essa può essere giudicata prominente da più di 6, 7, 8 o 9 persone. Infine, sono state selezionate le sillabe convergenti nell'80% dei casi come prominenti, che sono il 23, 56% del totale. È stato fatto poi un'altro tipo di selezione, in cui il peso più importante veniva considerato il livello di affidabilità dei giudici. In questo caso risultavano prominenti il 33, 46% delle sillabe nel corpus.

ii. Le features

Ogni sillaba all'interno dell'espressione ha un determinato numero di caratteristiche. Come è stato già accennato, ci saranno principalmente due attori a determinare le features: pitch-accent e force-accent. Senza andare troppo nel dettaglio, le caratteristiche di ogni sillabe sono le seguenti:

1. **durata nucelo**, normalizzata rispetto alla media e alla varianza del nucleo all'interno dell'espressione;
2. **spectral emphasis**, feature anch'essa normalizzta con `z-score`;
3. **movimenti del pitch**, feature calcolata a partire da parametri forniti dal modello `TILT` e con l'ausilio di specifici algoritmi per individuare il picco;
4. **intensità complessiva**;
5. **durata sillaba**, valore ricavato come per la durata del nucleo, ma riguardante l'intera sillaba.

A parte la durata della sillaba, tutte le features sono calcolate nel dominio del nucleo della sillaba. La comprensione di come queste caratteristiche si inseriscano nel quadro dello studio (durata della sillaba a parte), viene facilitata dalla gerarchia dei fenomeni proposta nel [1, Capitolo 3], e riproposta nella Tabella 1. Il fenomeno percettivo della prominenza è basato sui fenomeni prosodici dello *stress* e il *pitch accent*, a loro volta correlati a fenomeni acustici ricavabili da parametri fisici dell'enunciato. Inoltre, ogni fenomeno prosodico è sufficiente per individuare il fenomeno della prominenza, singolarmente o in presenza dell'altro.

iii. Risultati

Per le `PGM` è stato usato un training set di 100 espressioni mentre il test set ne conteneva 20. La Tabella 2 riporta le prestazioni raggiunte, senza (ricordiamo) l'utilizzo di alcuna caratteristica linguistica, solo tramite informazioni acustiche. I risultati ottenuti non possono essere comparati con altri studi nel settore, dato che sarebbe necessario avere lo stesso corpus.

In [2] è stato anche fatto vedere quanto le `PGM`, e in particolare le `LDCNF`, siano meglio in confronto alle *Support Vector Machines* (`SVM`). L'*F-measure* infatti migliora di mezzo punto, da 0.665 a 0.716 per il criterio della convergenza, e di 0.38 punti per il criterio *best-3*. Si cercherà di fare un confronto simile usando le reti neurali ricorsive, andando a vedere che miglioramento sono in grado di apportare alla causa.

IV. Reti Neurali Ricorsive

La Definizione 1 della prominenza classifica tale fenomeno come continuo e marcato rispettivamente al contensto che lo circonda. Questo significa che le tradizionali reti neurali non possono affrontare un problema del genere, non è chiaro infatti come possano analizzare per esempio, il significato di una parola quando esso dipende da quanto è stato precedentemente detto.

Le *Reti Neurali Ricorsive* (`RNN`) sembrano ovviare a questo problema. Le `RNN` sono un tipo di rete contenente al loro interno dei cicli, ciò rende possibile che le stesse informazioni persistano al loro interno.

Nella Figura 1 si può vedere come, preso un ciclo, una `RNN` si può interpretare come una successione di copie della stessa rete. La loro struttura *a catena* si adatta perfettamente all'elaborazione di sequenze e liste. L'interpretazione è la seguente: data un porzione di rete A, essa guarda l'input x_t e produce l'output h_t, permettendo tramite il loop di passare l'informazione dal passo t a $t+1$. Una `RNN` può quindi guardare dietro nel tempo, individuare una dipendenza fra passato e presente e quindi produrre l'output più opportuno. La domanda ora è la seguente: *quanto* indietro nel tempo? Sfortunatamente, le `RNN` perdono efficacia man mano che la distanza tra passato interessante e presente aumenta.

Tabella 1: *Gerarchia dei fenomeni coinvolti nel riconoscimento automatico della Prominenza*

Fenomeni percettivi	Prominenza			
Fenomeni prosodici	Stress		Pitch accent	
Fenomeni acustici	durata	enfasi spettrale	movimenti in F0	intensità globale

Tabella 2: *Il migliore sistema* `PGM` *(Latent-Dynamic Conditional Neural Fields) al variare del metodo di selezione delle sillabe prominenti.*

	Accuratezza AC	Precisione P	Richiamo R	F-score F1
Convergenza 80%	**0.875**	0.788	0.658	0.716
Best-3	0.855	**0.831**	**0.718**	**0.770**

i. Long Short Term Memory

Le *Long Short Term Memory* (`LSTM`) sono un particolare tipo di `RNN` capaci di apprendere dipendenze a lungo termine. Infatti, le `LSTM` sono state proposte proprio per ovviare al problema delle `RNN` precedentemente descritto. I moduli, in questo tipo di architettura (Figura 2), racchiudono al loro interno più layer con funzione di attivazione `sigmoid` e `tanh` che interagiscono fra loro in maniera ben specifica.

L'idea alla base è che la `LSTM` può rimuovere o aggiungere informazioni allo stato della sua cella. Tali scelte vengono influenzate da strutture chiamate *porta* (o *gate*). Una porta fa passare l'informazione nello stato della cella in maniera ottimale, dato che la porta stessa non è altro che uno strato di rete neurale con funzione di attivazione sigmoidea (cella σ in Figura 2). Si ricordi che il range di σ è compreso fra 0 (*non far entrare nulla*) e 1 (*fai entrare tutto*), è necessario ricordare anche che gli strati di tipo `tanh` restituiscono valori compresi fra 1 e -1. Seguono brevemente i passi che ci sono dietro al funzionamento di una `LSTM`, spiegando i principi sottostanti senza addentrarsi troppo nella loro complessità.

Informazioni da dimenticare All'arrivo dell'input x_t, la prima cosa da fare è decidere quale informazione precedentemente ottenuta va buttata via. Viene prelevato l'output h_{t-1} e l'input attuale x_t (Figura 2), e restituito il valore compreso fra 0 e 1 per ogni numero nello stato della cella C_{t-1}. L'equazione 5 esprime il calcolo dei relativi pesi e bias dello strato σ, con il dovuto output.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{5}$$

f_t è da associare alla parola inglese *forget*, letteralmente *dimentica al passo t*.

Decidere quali informazione assorbire Il prossimo passo è quello di decidere quali informazioni assorbire nello stato della cella. In questa fase è coinvolto un'altro strato σ, detto *porta di input*, e uno strato `tanh` che crea un vettore di valori candidati con i queli *potrebbe* essere aggiornato il nuovo stato. L'output dello strato σ è dato dall'equazione 6, mentre il calcolo dell'ipotetico stato aggiornato $\tilde{C}_t$ viene espresso dall'equazione 7.

$$i_t = \sigma(W_i \cdot [h_{t-1}, h_t] + b_i) \tag{6}$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, h_t] + b_C) \tag{7}$$

Aggiornamento dello stato Una volta deciso cosa si vuole ricavare dal nuovo input, è il momento di aggiornare effettivamente C_{t-1} per ottenere C_t.

1. per dimenticare quello che è stato deciso viene eseguito $C_{t-1} \cdot f_t$;
2. per assorbire quello che è stato ritenuto necessario viene eseguito $\tilde{C}_t \cdot i_t$;
3. i risultati dei due passi sopra vengono sommati (Equazione 8).

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{8}$$

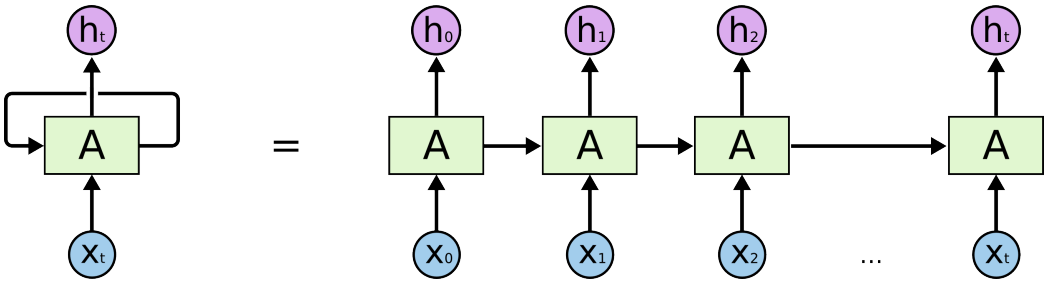


Figura 1: *Unrolling di una rete neurale ricorsiva*

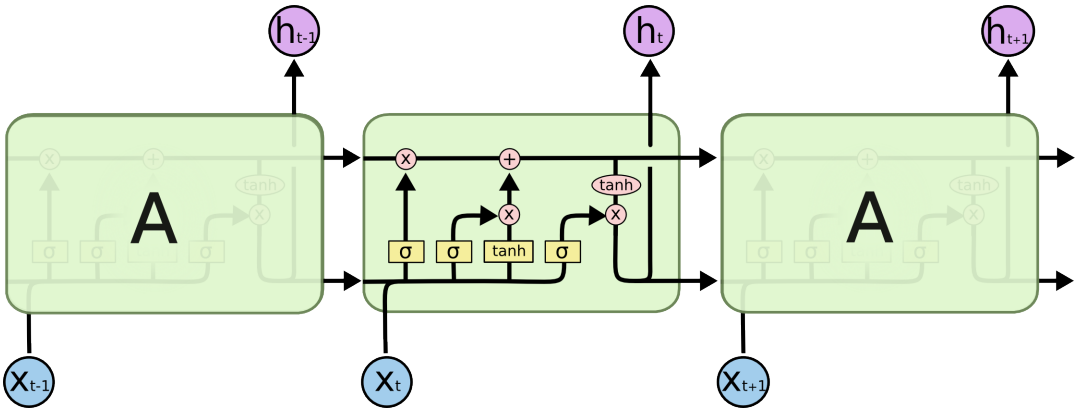


Figura 2: *Esempio di struttura interna alla base delle Long Short Term Memory (`LSTM`)*

Output Una volta creato lo stato C_t bisogna decidere l'output, esso sarà basato sempre sullo stato della cella, ma conterrà solo alcuni suoi valori filtrati. Ancora una volta, uno strato σ decide quali saranno le parti di $\tilde{C}_t$ a formare o_t. o_t verrà poi moltiplicato per lo stato C_t, filtrato da un *layer* `tanh`, ottenendo così l'output finale h_t, che verrà usato al passo t per ricavare h_{t+1}.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{9}$$

$$h_t = o_t * tanh(C_t) \tag{10}$$

La *mappa* dei parametri appena esposti si possono intuitivamente osservare nella Figura 3.

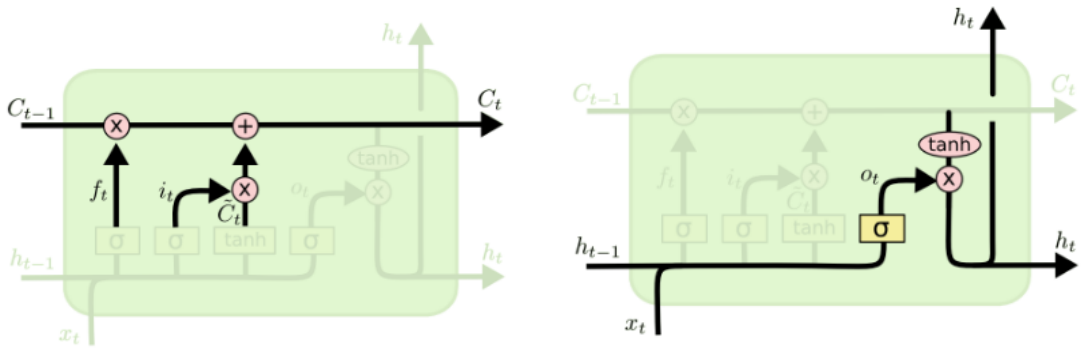
La breve spiegazione qui fornita riflette i concetti base[1], esistono comunque parecchie versioni concettuali che variano sensibilmente e ancora più numerose sono le implementazioni.

ii. Un problema di Sequenze

Il motivo per cui le `LSTM` sono adatte a trattare il riconoscimento automatico della prominenza è che sono in grado di elaborare *sequenze* di vettori[2], che rappresentano le *features*, e restituire sequenze di dimensione fissa a piacere. Ed è anche il motivo per il quale vengono usate nel trattamento dei video, testi, immagini, problemi `NLP`, e altro. I problemi sono risolvibili (classificazione, previsione, ecc...) tramite forme di mappatura rappresentati nella Figura 4. In base al tipo di mapping scelto, varia l'architettura della *Rete Neurale Ricorsiva*. Vedremo nella prossima sezione come risulta l'architettura nel nostro caso.

[1] Cristopher Olah, *Understanding LSTM Networks.*

[2] Andrej Karpathy, *The Unreasonable Effectiveness of Recurrent Neural Networks.*



(a) *Parametri calcolati nelle prime fasi* **(b)** *Parametri calcolati nelle ultime fasi*

Figura 3: *Mappa dei parametri interni a una* `LSTM`

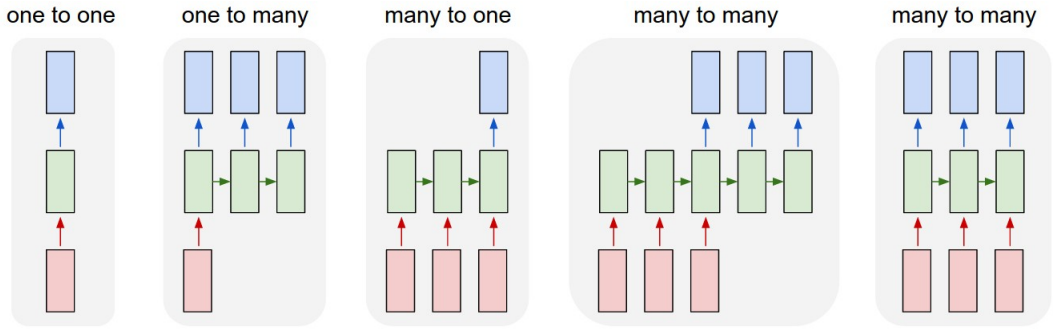


Figura 4: *Problemi sequence-to-sequence trattabili con le* `RNN`

V. Costruzione del Modello

Le `LSTM` precedentemente riportare, così come le reti neurali in genere, hanno diverse varianti e implementazioni. Per la realizzazione del modello è stato scelto `Keras` [4], che è un `API` ad alto livello per diverse tipologie di reti neurali. Il principio che sta alla base di `Keras` è che la chiave per fare della buona ricerca consiste nel rendere minima la distanza fra l'idea e la sua realizzazione, si ha quindi una libreria parecchio potente a disposizione. È scritta in `Python`, e si può scegliere l'implemtazione basata su `Theano` o `TensorFlow`.

In `Keras`, un modello è una pila di diversi strati (reti neurali) collegati fra loro, dove gli input entrano nello strato d'ingresso e fuoriescono risultati dallo strato di uscita. Si possono però unire diversi modelli in uno unico, combinando i loro output tramite gli strati di classe `Merge`.

Per la realizzazione di questo progetto è stato usato `Python 3.5` e `Keras 1.2.2`. Verranno ora descritti brevemente gli strati impiegati nel tentativo di costruire la miglior rete neurale.

Masking Layer Lo strato in questione ha il compito di *mascherare* i valori in input che sono uguali a un certo valore. Va tenuto presente che la rete neurale prenderà in input sempre frasi di lunghezza $n = 34$, me le espressioni più corte avranno (alla fine) vettori di zeri al posto delle features mancanti. In questo caso il valore da mascherare sarà quindi $\{0, 0, 0, 0, 0\}$:

```
model.add(Masking(
mask_value = [0.0, 0.0, 0.0, 0.0, 0.0]))
```

Significa letteralmente che la rete neurale non cercherà di imparare a partire dai valori mascherati.

Dense Layer È lo strato che implementa una tipica rete neurale densamente connessa, con un arbitrario numero di nodi, funzione di attivazione e altro.

LSTM Layer È l'implementazione di una rete `LSTM` descritta in i.

Dropout Layer L'obiettivo principale dello strato di `Dropout` è quello di prevenire l'overfitting, e viene spesso abbinato ai layer `LSTM`. Si tratta di una tecnica in cui nella fase di training un numero casuale di unità della rete vengono ignorate, costringendo i *neuroni* (o celle) attive a imparare anche per conto di quelli mancanti. L'effetto (desiderato) è che l'intera rete diventa meno sensibile agli specifici pesi di ogni unità di rete. Ogni volta, i nodi selezionati vengono gettati fuori dal quadro con una certa probabilità (per esempio il 50%).

```
model.add(Dropout(0.5))
```

TimeDistributed Wrapper Non si tratta di uno strato bensì di uno `wrapper`. Prende in input un determinato layer che viene applicato ad ogni intervallo di tempo dell'input della rete. Questo wrapper farà quindi sì che la rete restitiuisca sequenze.

```
model.add(TimeDistributed(Dense(5)))
```

i. Costruzione della rete

Avendo estratto le features di sillabe per ogni enunciato all'interno del corpus e le relative classificazioni di prominenza, viene individuata la frase contenente più sillabe in assoluto. Ogni sillaba possiede un vettore di 5 features e la sua classificazione binaria *prominente / non prominente*, e la frase più lunga contiene $n = 34$ sillabe. Si ha quindi il problema di mappare correttamente i seguenti insiemi:

$$\{v_1, ..., v_n\} \rightarrow \{p_1, ..., p_n\} \tag{11}$$

Nella Figura 4 ciò corrisponde al modello *many to many*, il quinto tipo. Sappiamo cosa è v_i, è necessario ora decidere cosa è p_i. All'inizio

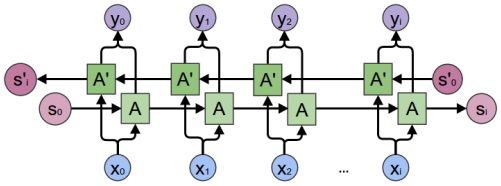


Figura 5: *Architettura astratta di una rete* `BLSTM`

l'assunzione era che $p_i \in \{0, 1\}$ per indicare la non prominenza e il contrario, ma tutti i modelli costruiti intorno a questa scelta erano davvero scadenti ($F1$ non più del 46%). Tali modelli prevedevano, oltre i dovuti layer `LSTM`, l'utilizzo di `Masking` come layer di input e uno strato `Dense(34)` che ritornasse 0 e 1 per ogni sillaba in ingresso.

Per cercare altre vie è stato asserito che $p_i \in \{\{0, 1\}, \{1, 0\}\}$, dove 1 nella prima posizione stava per indicare che la sillaba non è prominente e nella seconda posizione il contrario. Per realizzare la cosa è stato impostato `TimeDistributed(Dense(2))` come strato finale, con funzione di attivazione `softmax`, in modo che ritornasse dati in dimensione $(34, 2)$, con probabilità accumulata in una delle due posizioni. I sistemi risultanti davano un migliorato score $F1$ ma l'accuratezza era parecchio calata, a circa il 60%.

È stato infine intuito che la *sporcizia* è data dalle sillabe mancanti: per qualche motivo lo strato `Masking` non dava i risultati sperati, e le sillabe mancanti venivano denotate dalla rete come un'assoluta incertezza in cui alle entrambi posizioni veniva assegnata una probabilità del 50%. Il problema è stato raggirato togliendo tale strato e, decidendo di non ignorare le sillabe mancanti, ma di classificare pure quelle. L'output per ogni *timestep* è quindi diventato $p_i \in \{\{1, 0, 0\}, \{0, 1, 0\}, \{0, 0, 1\}\}$ (per indicare rispettivamente non prominenza, prominenza e assenza). Infinie, l'ultimo layer `Dense(3)` con attivazione `softmax` e dentro uno wrapper `TimeDistributed`, avrebbe concentrato le probabilità in una delle 3 posizioni.

Figura 6: *Costruzione del modello con* `Keras`

```
    numpy.random.shuffle(indexes)
    x_dataset = x_dataset[indexes]
    y_dataset = y_dataset[indexes]

    # Building Model
    model = Sequential()

    model.add(Bidirectional(LSTM(max_utterance_length / 2, return_sequences = True),
                            input_shape = (max_utterance_length, len(COLUMNS) - 1)))
    model.add(Dropout(0.5))

    model.add(TimeDistributed(Dense(len(prominent_syllable), activation = 'softmax')))

    model.compile(loss = 'binary_crossentropy', optimizer = 'adam',
                  metrics = ['accuracy', 'fmeasure', 'precision', 'recall'])

    model.fit(x_dataset[TRAIN_INDEXES],
              y_dataset[TRAIN_INDEXES],
              validation_data = (x_dataset[VALIDATION_INDEXES],
                                 y_dataset[VALIDATION_INDEXES]),
              nb_epoch = 50,
              batch_size = 2)

    metrics = model.evaluate(x_dataset[TESTSET_INDEXES],
                             y_dataset[TESTSET_INDEXES])
```

Tabella 3: *Risultati ottenuti in seguito al lavoro di progetto, confrontati con il modello* `LDCNF`.

Modello	Accuratezza A	Precisione P	Richiamo R	Fmeasure F1
LDCNF best-3	85.50%	83.10%	71.80%	77.00%
LSTM_17	91.99%	88.66%	87.13%	87.88%
LSTM_34	92.27%	88.84%	87.84%	88.33%
BLSTM_17x2	**94.28%**	**91.57%**	**91.25%**	**91.41%**

ii. Bidirectional `LSTM`

In base a prove empiriche la scelta finale è ricaduta sulla seguente architettura, in ordine:

- `LSTM` Layer;
- `Dropout` Layer;
- `Dense` layer contenuto dentro wrapper `TimeDistributed`.

In seguito ad alcune ricerche è stato scoperto che una versione di rete `LSTM`, ovvero le *Bidirectional* `LSTM` (`BLSTM`), si comportano molto bene nei problemi di *sequence-labeling* in ambito `NLP` (come nel seguente lavoro [5, Capitolo 5]). Sono state messe alla prova, incapuslando una rete `LSTM` dentro uno wrapper `Bidirectional` comodamente offerto da `Keras`. L'idea che sta alla base delle reti `BLSTM` è che un dato output y_t non dipende solo dalle esperienze passate ma anche da quelle $future$. Esse racchiudono infatti uno strato `LSTM` *forward* un'altro detto `backward` (Figura 5), creando una doppia dipendenza. Probabilmente non sempre un modello può guaradere *avanti*, ma in questo caso si può testare il comportamento di un'architettura del genere, in quanto una sillaba prominente lo è anche perché quelle dopo lo sono di meno. La struttura a pila è quindi diventata:

- `LSTM` contenuto dentro `Bidirectional`;
- `Dropout` Layer;
- `Dense` dentro `TimeDistributed`.

iii. Risultati

Venendo al dunque, delle 120 frasi a disposizione, il *training-set* ne contiene 85; il *validation-set* 15 e il *test-set* 20. Nei test riportati qui rientrano i 3 modelli che si sono comportati meglio in assoluto: `LSTM` a $n/2 = 17$ celle; `LSTM` a $n = 34$ celle; e infine una `BLSTM` con $n/2$ celle per ognuno dei due strati interni. Parte del codice è stato riportato nella Figura 6. Sono state provate diverse dimensioni di `batch` e numero di epoche, 50 epoche sono sembrate sufficienti in quanto troppe epoche portavano a fenomeni di *overfitting*.

Sono state realizzate 20 fasi di apprendimento, cambiando l'ordine delle espressioni nel corpus casualmente ogni volta ed estraendo alla fine le 4 metriche per poi farne la media. I risultati finali sono riportati nella Tabella 3.

VI. Conclusione

In questo lavoro di progetto, partendo da zero, è stato approfondito il fenomeno della prosodia nell'ambito *Natural Language Processing*. Una volta capiti i fenomeni sottostanti che la supportano, in una determinata gerarchia, è stato analizzato lo studio [2]. Allo scopo di ripetere lo stesso sperimento, usando lo stesso corpus, sono state studiate le *Reti Neurali Ricorsive*, e in particolare le *Long Short Term Memory*. Una volta costruito il modello ed eseguite le necessarie fasi di apprendimento, è stato fatto un procedimento analogo usando reti *Long Short Term Memory Bidirectional*, dove gli output diventano dipendenti anche dal *"futuro"*.

Concludendo si può dire che i modelli `PGM` sono stati superati di ben 10 punti per quanto riguarda la *Fmeasure*, tale risultato è dato dall'incremento signicativo della metrica *recall*, alzatasi di ben 15 punti. Si può anche dire che l'introduzione di dipendenze a doppio senso (`BLSTM`) introduce ulteriori miglioramenti.

Riferimenti bibliografici

[1] Fabio Tamburini, *Fenomeni Prosodici e Prominenza: Un Approccio Acustico*, 2005.

[2] Fabio Tamburini, Chiara Bertini, Pier Marco Bertinetto, *Prosodic prominence detection in Italian continuous speech using probabilistic graphical models*, 2014

[3] Fabio Tamburini, *Automatic Detection of Prosodic Prominence by Means of Acoustic Analyses*, 2015

[4] Chollet, François, *Keras*, `https://github.com/fchollet/keras`, 2015

[5] Alex Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, 2008