POLITECNICO DI TORINO

# Chatbot basato su RNN

Corso di Intelligenza Artificiale


Professore:
Elio Piccolo

Studenti:
Carlo Ventrella
Giuseppe Lillo

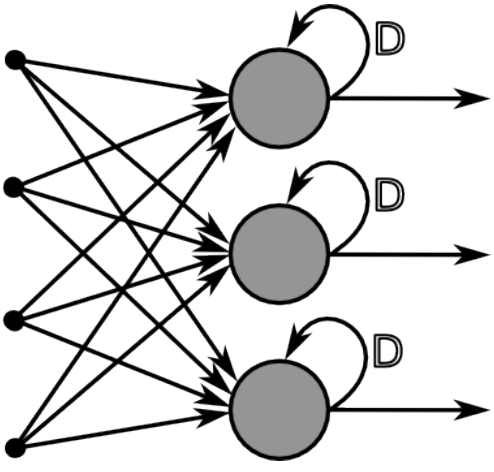

## Introduzione

Il chatbot è un agente intelligente ed interattivo in grado di colloquiare con un essere umano, e i progressi raggiunti finora ne hanno permesso l'integrazione in software di uso comune, dai noti assistenti vocali di Google e Apple, fino ad applicazioni di customer service per aziende.

Lo scopo di questa tesina è quello di analizzare e implementare un chatbot utilizzando un'architettura che sfrutta le Recurrent Neural Network per rispondere alle domande di un utente.

## Recurrent Neural Networks

Le Recurrent Neural Network (RNN) rappresentano un particolare modello di rete neurale contrapposto alle reti *feed-forward*, in cui gli output di ogni nodo rientrano come input degli stessi nodi.



Questa architettura permette di analizzare delle sequenze, la cui caratteristica principale è la dipendenza tra i vari elementi della sequenza: in una frase del linguaggio naturale del tipo *“Questo oggetto è mio”*, l’aggettivo *questo* è in stretta relazione con *oggetto*, così come *mio*.

Una rete di tipo feed-forward sarebbe costretta ad analizzare contemporaneamente tutte le parole di una stessa frase, e quindi non riuscirebbe ad apprezzare correttamente le dipendenze tra di esse.

Una RNN invece permette di analizzare i singoli elementi di una sequenza, uno per ogni *timestep*, mantenendo uno ***stato*** tra le varie iterazioni che contiene informazioni su quanto è accaduto fino a quel momento, cioè dal timestep 0 al *t*.

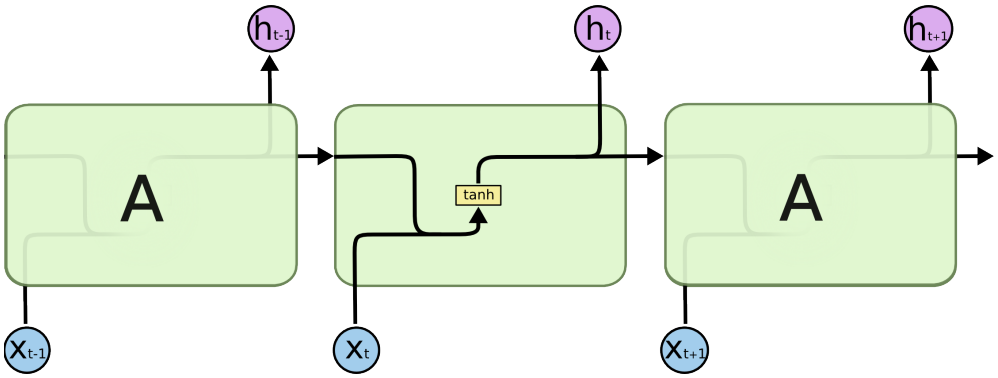
In una RNN standard lo stato è dato semplicemente dall’output del nodo.

## Long short-term memory (LSTM)

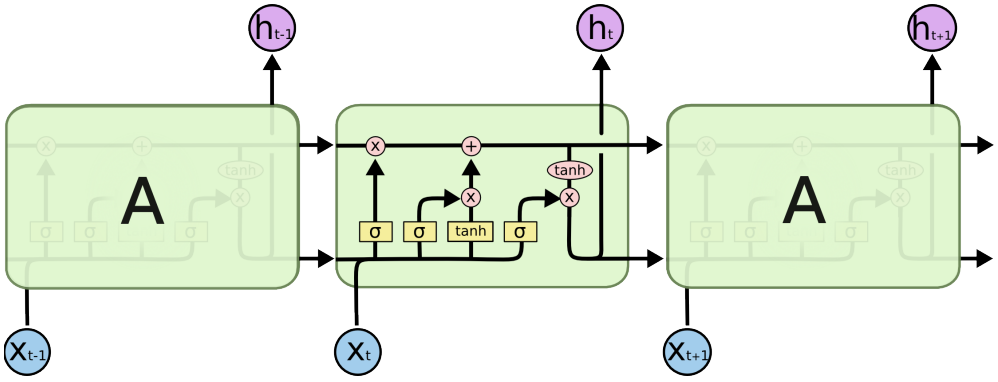
Nonostante le RNN siano state pensate per l’analisi di sequenze, mostrano dei difetti, come nel caso di sequenze di input troppo lunghe. Questo perchè lo stato memorizzato tra i vari timestep non è in grado mantenere le informazioni per troppo tempo.

Per risolvere questo problema, nel 1997 Hochreiter e Schmidhuber hanno proposto un particolare tipo di nodo ricorrente: il nodo LSTM.

Il nome deriva dal fatto che questo tipo di nodo è in grado di modellare la *memoria a breve termine* (short-term memory) per un lungo periodo di tempo.




*Un nodo di una RNN srotolato nel tempo.*




*Un nodo LSTM srotolato nel tempo.*

L'idea di base è di utilizzare uno stato $C$ oltre all'uscita $h$, in cui è possibile aggiungere o rimuovere informazioni derivanti dai timestep precedenti: questo potrebbe essere usato ad esempio per riferirsi a più soggetti nel corso dell'analisi di una frase, o di ricordare il soggetto della frase, in modo tale da essere in grado di selezionare i pronomi adeguati.

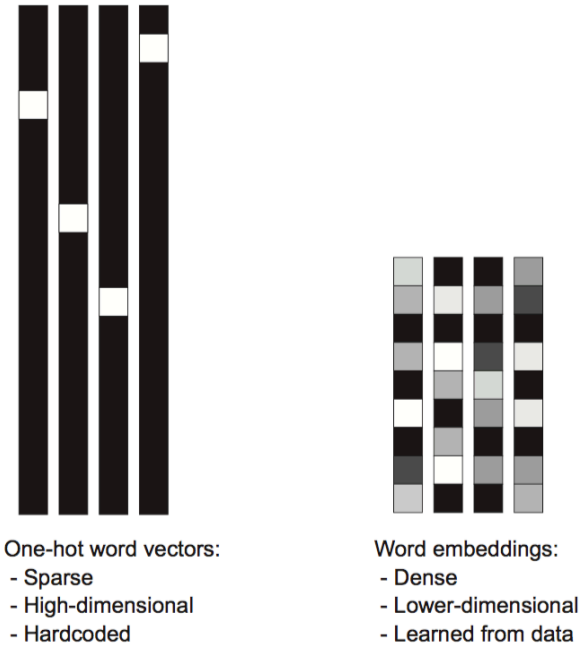
## Word Embedding

Le RNN, come ogni altro tipo di rete neurale, richiedono un input di tipo numerico. Diversi approcci per trattare input testuali consistono nel dividere il testo in *token* e poi associare a questi dei vettori.

I token possono essere caratteri, parole, o *n-grams* (gruppi sovrapposti di *n* parole consecutive).

L'approccio più comune è quello di dividere l'input in parole, e di codificarle con la tecnica del *one-hot encoding*: ogni parola è associata a un vettore sparso di lunghezza N (dimensione del vocabolario), in cui l'unica cella attivata (impostata a 1) corrisponde a quella con indice pari alla parola in questione. È un approccio che ha come chiaro limite la grandezza del dizionario.

La tecnica utilizzata in questo caso prende il nome di *word embedding*, e consiste nell'associare ad ogni parola un vettore di piccola lunghezza e denso. Rispetto al one-hot encoding, nel word embedding i vettori sono sono ricavati dai dati, tramite allenamento di reti neurali, algoritmi di riduzione della dimensionalità o modelli probabilistici.




*Differenze one-hot word vector e word embedding.*

Da un punto di vista geometrico i word embedding mappano le parole in uno spazio N dimensionale, dove N è la dimensione dei vettori. I vettori non vengono assegnati alle parole in maniera casuale, ma vengono generati e allenati in modo tale che la relazione tra due vettori rifletta la relazione semantica che lega le parole a cui questi sono associati.

Ci si aspetta quindi che parole simili, sinonimi per esempio, siano associati a vettori simili, e che semplici trasformazioni geometriche abbiano un significato nel linguaggio umano.

Un classico esempio sono le trasformazioni di genere e numero:

$$\text{king} + \text{female} = \text{queen}$$

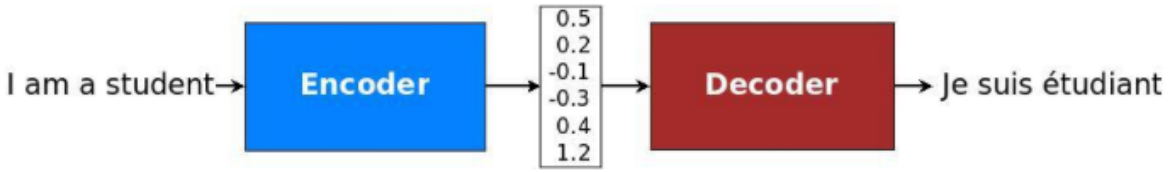
$$\text{king} + \text{plural} = \text{kings}$$

Poiché la generazione dei word embedding è computazionalmente onerosa, spesso si preferisce utilizzare word embedding *pre-allenati*: tra i più famosi si citano GloVe, Word2Vec e FastText.

## Sequence-to-Sequence

Alla base di un chatbot, di un traduttore, o di un modello che riassume un testo, vi è il problema di come passare da un input di una certa lunghezza ad un output di un'altra lunghezza.

Nel campo della Neural Machine Translation (NMT), un approccio tradizionale era quello di dividere il testo in chunk e tradurli separatamente. In questo modo la traduzione non sempre è efficiente, in modo particolare per input lunghi, perchè non rispecchia il nostro modo di tradurre una frase: l'approccio di un essere umano è quello di leggere una frase, capirla, e tradurla, ed è quello che i modelli sequence-to-sequence (*seq2seq*) cercano di emulare.

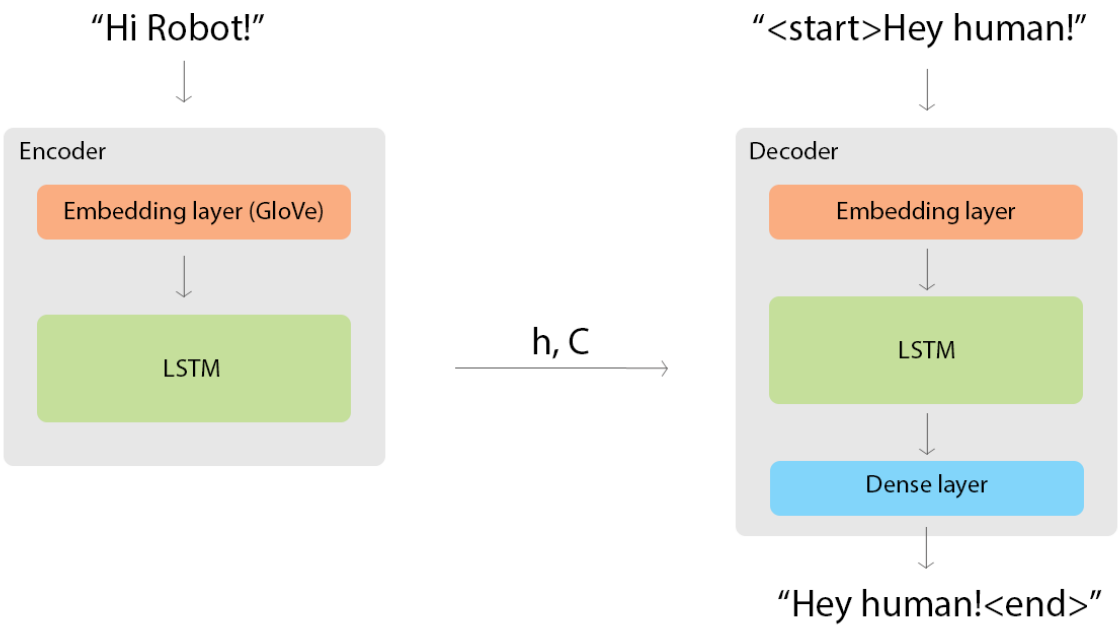



*Architettura encoder-decoder.*

I modelli seq2seq sono caratterizzati da una architettura encoder-decoder, in cui l'encoder riceve una frase e produce uno stato, nella forma di uno o più vettori, che ne rappresenta il significato.

Il decoder riceve in input questo stato, e produce un risultato.

Questa architettura risolve il problema della traduzione locale, e permette di tenere conto delle dipendenze semantiche e grammaticali "a lungo raggio" che caratterizzano le nostre frasi.




*Architettura seq2seq implementata.*

## Architettura

Per la loro struttura, le LSTM risultano essere una scelta naturale nella costruzione di encoder e decoder.

L’encoder è caratterizzato da un livello di embedding, in cui gli input (le parole), vengono associate ai rispettivi vettori. Gli embedding vengono passati in input alla rete LSTM, caratterizzata da un unico livello, che produce uno stato associato alla frase in input.
Lo stato, come anticipato, è composto dai vettori $C$ e $h$.

L’architettura scelta per questo progetto prevede l’utilizzo di embedding pre-allenati, si è scelto di utilizzare GloVe a tal proposito, con word embedding di 100 elementi.

Il decoder presenta una architettura analoga a quella dell’encoder, tuttavia in questo caso gli embedding saranno calcolati durante la fase di training del modello. Inoltre, il decoder presenta un terzo livello, cioè un livello di output composto da M nodi, (dove M è il numero di parole presenti nel vocabolario) e con funzione di attivazione *softmax*.

$$\text{softmax}(z)_j = \frac{e^{(z)_j}}{\sum_{m=1}^{M} e_m^z} \quad j = 1...K$$

La funzione *softmax* è comunemente usata nei layer di output delle reti neurali, in quanto mappa un vettore M-dimensionale di Z valori reali arbitrari, in un vettore M-dimensionale di valori compresi tra 0 e 1, la cui somma è 1. La softmax assegna quindi ad ogni nodo, cioè ad ogni parola, una probabilità.

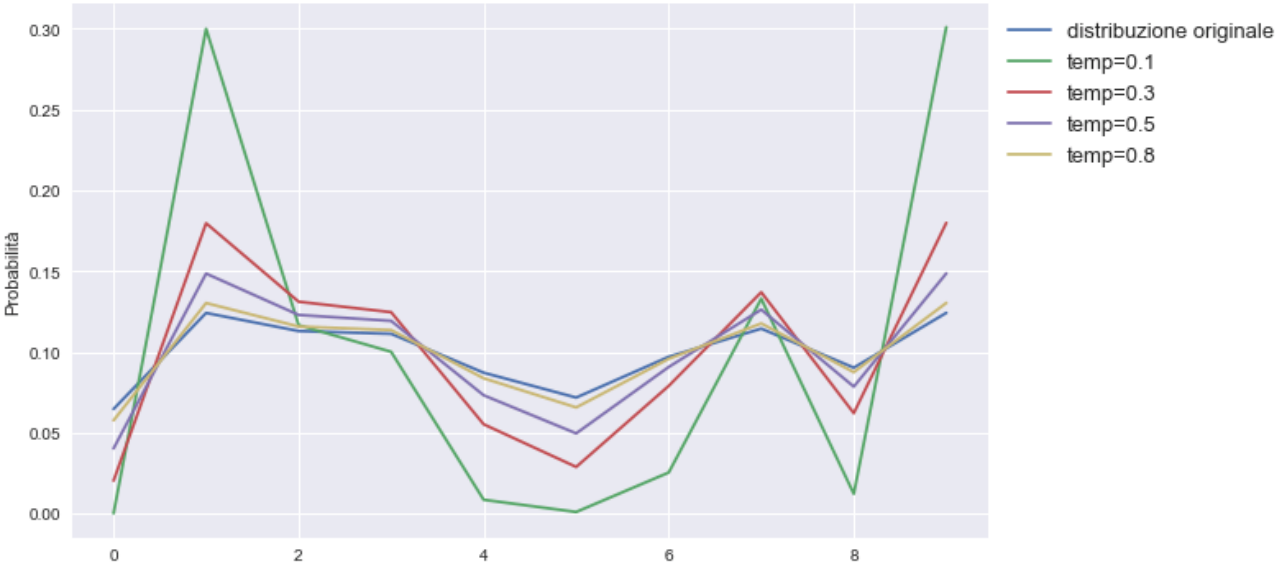
La scelta più naturale potrebbe essere quella di selezionare la parola che presenta la più alta probabilità di essere estratta (*approccio greedy*), ma questo significa anche che dato

un contesto, e dato un input, il decoder sarà sempre portato a selezionare la stessa parola.

Per introdurre una maggiore casualità nella selezione della parola si opera sulla *softmax temperature*, un parametro che ha il compito di generare una nuova distribuzione di probabilità più o meno uniforme rispetto a quella originale.

Aumentando la temperatura si rende la distribuzione più *uniforme*, fino a ricondurci alla distribuzione originale che coincide con temp=1.

A questo punto, tenendo conto della nuova distribuzione di probabilità, si estrae una parola in maniera casuale.




*Andamento della distribuzione di probabilità al variare della softmax temperature.*

## Training

Nella fase di training l'encoder riceve in input il vettore che contiene gli indici delle parole presenti nella frase. Il vettore ha una lunghezza pari alla lunghezza massima che si è scelto per le frasi. Nel caso in esame è 40.

Es. "Ciao robot!" sarà tradotto in un vettore lungo 40, e sarà del tipo [12, 456, 23, 0 … 0],

dove i primi tre numeri sono gli indici delle tre parole, "ciao", "robot" e "!". I restanti 37 elementi sono lasciati a 0.

Gli input vengono passati al livello di embedding, che traduce gli indici in embedding.

Come anticipato, gli embedding dell'encoder sono stati ottenuti da GloVe, e non saranno perciò allenati ulteriormente.

Gli embedding passano alla LSTM dell'encoder, che produrrà i vettori $C$ e $h$. I pesi di questa RNN saranno aggiustati durante la fase di back-propagation.

A questo punto l'attenzione si sposta sul decoder, che si allena a predire la frase di input, ma traslata di 1 timestep: il decoder, a partire dalla parola di *[start]* cercherà di predire la parola successiva, fino ad arrivare alla parola *[end]*.

Nel caso in cui la parola generata sia diversa da quella target, sarà proprio quella esatta a rientrare come input nella LSTM per continuare a generare la frase, anzichè quella predetta in maniera errata. In questo modo il modello viene reso più stabile e l'apprendimento risulta più veloce. Questo metodo è noto in letteratura con il nome di *teacher forcing*.

Il decoder riceve in input la frase da predire, che, con le stesse modalità di prima, deve passare da un livello di embedding. In questo caso gli embedding non sono pre-allenati, ma sono inizializzati in maniera casuale e poi aggiustati durante il training.

La LSTM del decoder riceve in input questi embedding, e il suo stato è inizializzato con la coppia $(C, h)$ prodotta dalla LSTM dell'encoder.

Nella fase di training questa LSTM ha il compito predire l'input del decoder traslato di 1.

A questo punto dalla differenza tra l'output della LSTM e l'input del decoder può partire la fase di back-propagation, che comporterà l'aggiustamento dei pesi delle due LSTM e del livello di embedding del decoder.

Il dataset di training è composto da 20.000 domande/risposte, ciascuna frase contenente massimo 50 parole.
Il training è stato effettuato con un batch size di 120 per un totale di 500 epoche utilizzando un ottimizzatore *RMSprop* (una variante del Gradient Descent con learning rate variabile), impiegando un totale di 8 ore e mezza su una GPU Nvidia P4000.

## Inference

Dopo che la rete è stata allentata si passa alla fase di inference, quella in cui si utilizzano input sconosciuti.

Dal punto di vista dell'encoder non ci sono cambiamenti: come prima ha il compito di generare uno stato, utilizzando questa volta i pesi ottenuti dall'allenamento della rete.

Il decoder agisce invece diversamente: prima l'input del layer di embedding era la frase da predire traslata di 1, nella fase di inference invece l'input corrisponde (in un primo momento) all'identificatore di inizio sequenza.

L'input viene quindi tradotto in embedding, e passato alla LSTM del decoder, che anche in questo caso è inizializzata con lo stato della LSTM dell'encoder.

La LSTM del decoder, basandosi sullo stato e sull'input del decoder, estrae la parola successiva basandosi sulla distribuzione di probabilità prodotta dalla rete, e la utilizza come input del decoder per predire la parola successiva. Il processo continua fin quando o la lunghezza massima è stata raggiunta, o è stato selezionato l'identificatore di fine sequenza.

# Dataset

Titolo:  Cornell Movie-Dialogs Corpus

URL: http://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html

Il dataset contiene conversazioni estratte da copioni di 617 film, per un totale di 304 713 espressioni, per un totale di 83096 conversazioni.

Dividendo ogni conversazioni una lista di botta-risposta, si sono ottenute 221617 coppie.

Data la grandezza del dataset e della limitata potenza computazionale a disposizione, si è scelto di utilizzare solo le frasi più corte di 50 parole, scendendo così a 93100 coppie, di cui sono state selezionate le prime 20000.

## Preparazione del dataset

Prima di poter utilizzare il dataset è stato necessario ripulirlo. Le espressioni spesso contengono caratteri che sporcano il testo e che è necessario eliminare.

Es.

> So -- which Dakota you from?
>
> -- no -- Marie -- <u>no</u>!
>
> "Will you hear ""Last Post"", Sir?”

Inoltre, poiché il dataset è costruito su copioni di film, ci sono spesso riferimenti ai nomi dei personaggi, che si è ritenuto opportuno sostituire con un opportuno tag.

Anche i numeri sono stati sostituiti da un tag.

Infine, si è cercato di unificare espressioni come “won’t” e “will not”, oppure “am” e “'m".

Dopo aver ripulito il dataset, il risultato è il seguente.

| Domanda | Risposta |
|---|---|
| C'esc ma tete. This is my head | Right. See? You're ready for the quiz. |
| You have my word. As a gentleman | You're sweet. |
| Hi. | Looks like things worked out tonight, huh? |
| Have fun tonight? | Tons |
| Gosh, if only we could find Kat a boyfriend… | Let me see what I can do. |
| C'esc ma tete. This is my head | Right.  See?  You're ready for the quiz. |
| Where did he go? He was just here. | Who? |

| Domanda | Risposta |
| --- | --- |
| Where’ve you been? | Nowhere… Hi, Daddy. |

## Output

*hi*
> *hi*

*how are you?*
> *i am sure it was a long time .*

*are you happy?*
> *yeah .*

*are you sad?*
> *yeah .*

*what's your number?*
> *you know what i am .*

*no i don't know*
> *you will not*

*really?*
> *yeah . why not sorry .*

do you like pizza?
> no .

do you like computers?
Bot:I'm sorry human,"computers" is not in my dictionary, but it is 87.5198297441% similar to "computer"
> no .

where are you going?
> i am going out to work with two . see you long .

who are they?
> i do not know .

how many hours do you sleep usually?
> two weeks .

*Alcuni esempi di risposte del chatbot con T=0,2.*

## Conclusioni

Il chatbot prodotto è in grado di rispondere con frasi di senso compiuto in quasi tutti i casi. Vi sono evidenti problemi di risposte troppo legate al contesto; in questo senso il dataset scelto risulta non propriamente adatto al training di un chatbot.
Un dataset di sole 20000 frasi risulta troppo scarno per poter allenare un agente che deve rispondere in maniera generica a qualsiasi frase.

La scelta di questi dati è stata dettata dalle poche risorse computazionali a disposizione: per poter creare un chatbot allo stato dell’arte ci sarebbe bisogno di molto più tempo e risorse.
Esistono inoltre meccanismi più complessi che permettono di migliorare in modo consistente il modello. Uno dei più usati è il meccanismo di *Attention*, che permette ad una RNN di focalizzarsi su determinati input anzichè su tutti quanti.
Nei modelli allo stato dell’arte l’Attention è diventato di fatto uno *standard de facto*, presente in tutti i modelli orientati all’analisi di sequenze.
All’interno del chatbot realizzato l’Attention non è stato utilizzato per questioni di tempo e complessità di implementazione.