Predictive Word System

Emanuele Mirante

June, 2022

Abstract

La previsione del testo durante la digitazione di un determinato utente consente di risparmiare molto tempo grazie alla comprensione dei modelli di testo dell'utente. Il lavoro mira ad ottenere un semplice "predictive word system" studiando i "pattern" ricavati da "I promessi sposi" di Alessandro Manzoni.

1. Predictive word system

1.1. Introduzione

Il "predictive word system" è un sistema che facilita la digitazione su un dispositivo o un motore di ricerca, suggerendo le parole che l'utente finale potrebbe voler inserire in un campo di testo. Le previsioni si basano sul contesto di altri messaggi o ricerche fatte dall'utente e sulle prime lettere digitate.È usato quotidianamente quando si scrivono messaggi, e-mail o si fanno ricerche sul web:in figura 1 sono mostrati gli esempi di "scrittura intelligente" per Google, Gmail e WhatsApp.

Nelle successive sottosezioni sarà presentata una panoramica delle reti "RNN" e delle LSTM, successivamente sarà descritta la "pipeline" di lavoro seguita, dal processamento dei dati all'implementazione del modello.

1.2. RNN

Una rete neurale ricorrente (RNN) è una rete neurale costituita da uno stato nascosto **h** ed un output y, che opera su una sequenza di valori a lunghezza variabile $x = (x_1, ..., x_T)$. É chiamata ricorrente perché presenta una connessione retrograda, che consente alle informazioni contestuali di passare attraverso la rete, in modo che gli output rilevanti delle fasi temporali precedenti possano essere applicati alle operazioni di rete nella fase temporale corrente. Ad ogni time step t, l "hidden state" h_t é aggiornato tramite:

$$h_t = f(h_{t-1}, x_t),$$

dove f è una funzione di attivazione non-lineare. Contrariamente a una rete neurale semplice,quindi, nelle reti neurali ricorrenti l'output non dipende solo dall'input corrente ma anche dallo stato del sistema.

Uno dei motivi per utilizzare gli RNN è il vantaggio di ricordare le informazioni passate. Tuttavia, potrebbe non riuscire a memorizzare le informazioni di molto tempo addietro.[**1**]

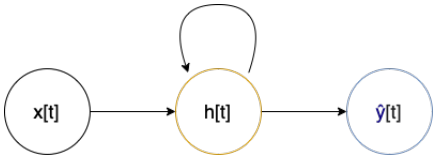


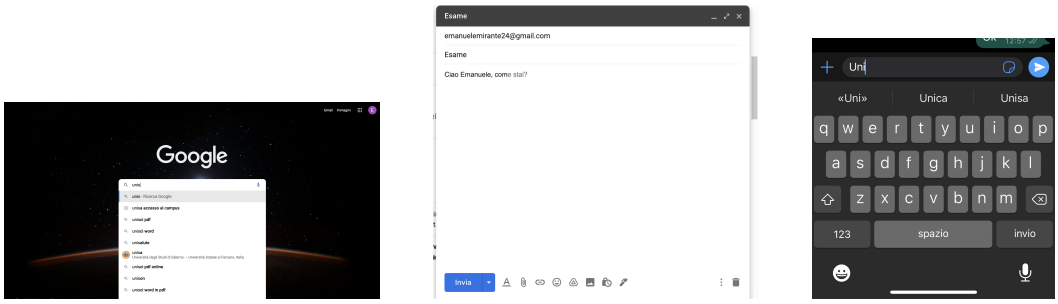
Figure 2: Architettura RNN

1.3. LSTM

Le reti RNN soffrono del problema della scomparsa/esplosione del gradiente e non riescono a memorizzare gli stati per un periodo temporale molto lungo.L'architettura LSTM tenta di risolvere questi problemi, un livello LSTM è costituito da un insieme di blocchi ricorrenti collegati, noti come blocchi di memoria. Ognuno contiene una o più celle di memoria collegate in modo ricorrente e tre unità moltiplicative: un "input gate"(che decide quali valori dall'input devono essere utilizzati per aggiornare lo stato di memoria), un "forget gate" (che decide quali informazioni devono essere scartate) e un "output gate" (che decide cosa produrre in base all'input e alla cella di memoria). Le equazioni delle celle LSTM in un dato istante temporale t sono:

- $F_t = \sigma(W_F x_t + U_F h_{t-1} + b_F)$,
- $I_t = \sigma(W_I x_t + U_I h_{t-1} + b_I)$,
- $M_t = tanh(W_M x_t + U_M h_{t-1} + b_M)$,
- $O_t = \sigma(W_O x_t + U_O h_{t-1} + b_O)$,
- $C_t = F_t \; o \; C_{t-1} + I_t \; o \; M_t$,
- $h_t = O_t \; o \; tanh(C_t)$

Dove $F_t, I_t \; ed \; O_t$ sono rispettivamente il "forget gate", l'"input gate" e l'"output gate"; M_t è un vettore di attivazione della cella di input; C_t è lo status della cella;h_t è il vettore di output della cella LSTM; x_t è il vettore di input della cella LSTM;



(a) Google search (b) Gmail (c) WhatsApp

Figure 1: Tipi di "Predictive word"

W, U *e* b sono rispettivamente le matrici dei pesi ed il vettore dei bias da regolare durante il processo di addestramento; σ è la funzione di attivazione sigmoidea.[**2**]

1.4. Presentazione e processamento dei dati

Come anticipato nell'introduzione, l'obiettivo è quello di generare una parola dato l'input dell'utente (e.g figura 1).Per fare ciò la rete ha bisogno di un "corpus" da cui imparare.Il testo scelto è stato "I promessi Sposi" pubblicato nel 1840 da Alessandro Manzoni, in particolare sono stati utilizzati i primi 30 capitoli del libro recuperati dal sito online "Project Gutenberg".[1] È stato preferito questo testo perché:

- è abbastanza grande da permettere alla rete di "imparare" al meglio(comprende 163.502 parole e 1.008.822 di caratteri),
- è scritto con un linguaggio particolare e riconoscibile(il fiorentino parlato) e ciò permette di valutare se il modello è capace di riprodurre uno stile di scrittura simile.

Una volta caricato il testo, questo è stato processato in modo da far corrispondere ad ogni carattere presente un proprio valore numerico(Fig. 3).Successivamente il testo è stato codificato in base a questi valori.

Esempio: se si hanno i seguenti indici$[q:0, u:2, e:7, l:1, '':10, r:9, a:3, m:6, o:20]$,la stringa: "*quel ramo*" verrà codificata come [0, 2, 7, 1, 10, 9, 3, 6, 20]

```
{'a': 0,
 'b': 1,
 'c': 2,
 'd': 3,
 'e': 4,
 'f': 5,
 'g': 6,
 'h': 7,
```

Figure 3: Indici dei primi otto caratteri

[1] https://www.gutenberg.org/

L'intento è che il modello preveda il successivo carattere data una sequenza storica di caratteri.La lunghezza della sequenza è stata impostata a 40 caratteri, suddividendo il testo in 25220 sentenze di 40 caratteri ciascuno.Successivamente le sentenze sono state trasformate affinché si abbia una sequenza di input associata ad una sequenza di "target" creata spostando la sequenza di input di un passo avanti.

Esempio:
sequenza di input: "quel ram"
sequenza target:"uel ramo"

1.5. Modellazione

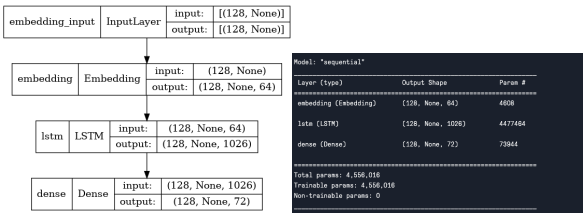


Figure 4: Architettura Modello

L'architettura è stata costruita sequenzialmente su python con l'utilizzo di TensorFlow, in particolare sono stati implementati tre strati:

- Un "Embedding layer",

  lo strato di "Embedding" fungerà da livello di input mappando gli indici numerici di ciascun carattere su un vettore con un numero di dimensioni pari a 64. Il numero di pesi sarà quindi uguale alla lunghezza del vocabolario usato ,ovvero 72, moltiplicato il numero di dimensioni "Embedding".

- Un "layer LSTM",

  lo strato LSTM presenta un numero di "hidden units" pari a 1026, tenendo presente le

equazioni nella sottosezione 1.3, il numero di pesi da stimare è uguale a: $(numero\ di\ input = 1026 + dimensione\ input = 64 + bias = 1) * 1026 * 4$.

- Un "Dense layer",

  il layer di output infine, grazie all'utilizzo della funzione di attivazione "softmax",è stato utilizzato per poter ricevere in output un vettore di probabilità di un numero pari alla dimensione del vocabolario. Il numero di pesi è calcolato come $(numero\ di\ input = 1026 + bias = 1) * (numero\ di\ output = 72)$

Il modello è stato addestrato per 100 epoche utilizzando la GPU(30 GiB RAM — 8 CPUs) messa a disposizione gratuitamente da "Paperspace"[2] e come dimensione del "batch" è stato scelto 128.Come suggerito dal sito ufficiale di TensorFlow per un problema simile[3], è stata utilizzata la "Sparse categorical cross entropy loss" come funzione di perdita, mentre per aggiornare i pesi della rete in modo iterativo è stato utilizzato il metodo di "Adam".

Per la valutazione del modello è stata presa in considerazione la generazione del testo ogni 10 epoche: in figura 5 ci sono alcune delle previsioni avvenute dopo le prime dieci epoche, mentre in figura 6 ci sono alcune delle generazioni avvenute dopo 100 epoche.

```
Stringa di input: ['lu']
 Previsione: ['lucr', 'ludde']
Stringa di input: ['don ab']
 Previsione: ['don abbia', 'don abbastanza']
Stringa di input: ['monaca di ']
 Previsione: ['monaca di non sape nulda, pe ess in quel che ']
```

Figure 5: Testo generato dopo 10 epoche

```
Stringa di input: ['lu']
 Previsione: ['lucia', 'luce']
Stringa di input: ['don ab']
 Previsione: ['don abbondio', 'don abbondio']
Stringa di input: ['monaca di ']
 Previsione: ['monaca di chi ingegnero i posteri di gente arciso']
```

Figure 6: Testo generato dopo 100 epoche

Nelle prime due previsioni si è voluto valutare il comportamento del modello nel completamento di un input dato dall'utente mimando quello che avviene nelle chat di WhatsApp (Figura 1), nell'ultima invece si è voluto valutare il comportamento del modello nella generazione di un'intera frase. Come si nota dalle immagini, le previsioni fatte a completamento delle 100 epoche sono nettamente migliori, ad esempio all'input "don ab" il modello finale è riuscito a catturare il pattern presente nel testo dato come input, mentre il modello "allenato" per sole 10 epoche pur generando parole sensate(si ricorda infatti che il modello prevede un carattere alla volta) ha accoppiato a "don" due parole che nel testo originale non sono precedute da esso. Nella generazione della frase in figura 6, il linguaggio particolare del fiorentino ottocentesco sembra essere stato ben riprodotto inventando anche una parola non presente nel testo("arciso") il cui suono appare essere coerente con le parole in uso all'epoca.

[2] https://www.paperspace.com

[3] https://www.tensorflow.org/text/tutorials/textgeneration

References

[1] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation".Doi:10.3115/v1/d14-1179