

## MIGLIORAMENTI

Per migliorare il nostro LM, bisogna approfondire il contributo che il contesto fa alla precisione delle parole corrette.

## PROBLEMI N-GRAM

- 1) La dimensione di  $\hat{W} \propto V \times V \times V \times \dots$ ,  
lunghezza contesto
- 2) Sparsezza del corpus.

Nel 2003, Bengio et al, introdussero il primo **large scale deep learning model** per i LM. Questo modello riesce a rispondere alle seguenti due domande:

- 1) Possiamo imparare la rappresentazione delle parole, tenendo conto anche di tutti gli n-gram "simili" che ci sono nel training corpus?
- 2) Possiamo imparare (e salvare) la densità di probabilità in una rete neurale?

## FORWARD- FORWARD LANGUAGE MODEL

$$P[w_i | w_{i-1} \dots w_{i-n-1}] = f(w_i | w_{i-1} \dots w_{i-n-1})$$

Dove  $f$  è una funzione del tipo  $f: V \rightarrow \mathbb{R}$ . Ovvero una funzione che associa una parola alla sua probabilità.

Possiamo decomporre  $f$ , come segue:

$$f(w_i | w_{i-1} \dots w_{i-n-1}) = g(w_i, c(w_{i-1}), \dots, c(w_{i-n-1}))$$

Così:

1)  $c$  è l'**embedding function** che mappa le parole  $w \in V$  alle relative rappresentazioni ( $\mathbb{R}^{d \times 1}$ )

$$c: V \rightarrow \mathbb{R}^{1 \times d}$$

2)  $g$  invece associa le rappresentazioni alle probabilità

$$g: V \times \mathbb{R}^{(n-1) \times d} \rightarrow \mathbb{R}$$

Possiamo raggruppare le parole nell'**embedding matrix**  $C$ :

$$C = \left[ \begin{array}{c|c} \begin{matrix} \xleftarrow{ } c(w_1) \xrightarrow{ } \\ \xleftarrow{ } c(w_2) \xrightarrow{ } \\ \vdots \\ \xleftarrow{ } c(w_n) \xrightarrow{ } \end{matrix} & \mid \\ \hline & \mathbf{v} \in \mathbb{R}^{V \times d} \end{array} \right]$$

Se rappresentiamo le parole con i **one-hot vector**, abbiamo:

$$c(x_k) = x_k C$$

La matrice  $C$  contiene  $d \cdot V$  elementi, che saranno imparati dalla rete (per poi essere confrontati con  $\hat{W}$ ).

Costruiamo ora l'ingresso di  $g(\cdot)$ , concatenando orizzontalmente gli **M-1 word embeddings**:

$$1 \quad X = \left[ \begin{array}{c|c|c} c(x_{i-1}) & \dots & c(x_{i-m+1}) \end{array} \right] \quad | \quad 1$$

$(M-1) \times d$

Anche  $g$  è imparata dalla rete come segue:

$$2 \quad \begin{aligned} y &= \tanh(x W_1) \\ g &= x W_3 + y W_2 \end{aligned}$$

Qui la funzione  $f$  è calcolata (e non imparata) con la softmax:

$$3 \quad f(x_i, \dots, x_{i-m+1}) = \text{softmax}_i(g)$$

1, 2, 3 formano il FFLM.

Ricapitolando abbiamo:

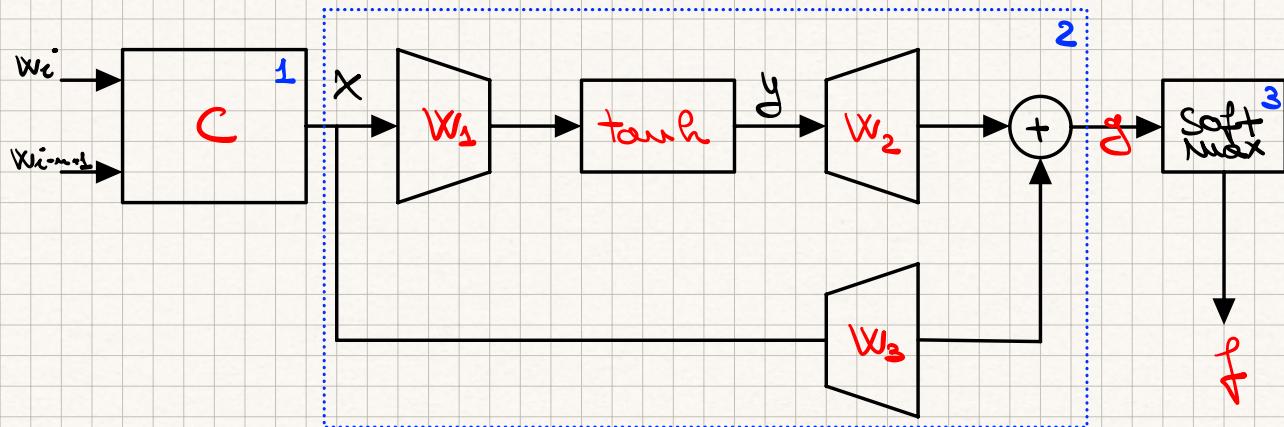
$$1) \quad x = [c(x_{i-1}) | \dots | c(x_{i-m+1})]$$

$$2) \quad y = \tanh(xW_1)$$

$$g = xW_3 + yW_2$$

$$3) \quad f(x_i, \dots, x_{i-m+1}) = \text{soft-max}_i(g)$$

## SCHESMA FFLM



Sia  $h$  il numero di hidden units in  $W_1$ , mentre  $d$  la grandezza dei word embeddings:

$$C \in \mathbb{R}^{V \times d}$$

$$x \in \mathbb{R}^{1 \times (m-1) \cdot d}$$

$$W_1 \in \mathbb{R}^{(m-1) \cdot d \times h}$$

$$W_2 \in \mathbb{R}^{h \times V}$$

$$y \in \mathbb{R}^{1 \times h}$$

$$W_3 \in \mathbb{R}^{(m-1) \cdot d \times V}$$

$$W_3 \in \mathbb{R}^{(m-1) \cdot d \times V}$$

$$g \in \mathbb{R}^{1 \times V}$$

$$f \in \mathbb{R}$$

Dove  $C$ ,  $W_1$ ,  $W_2$  e  $W_3$  sono i parametri della NN. La complessità totale è  $O(V \cdot d \cdot m)$ , lineare.

N-gram aveva complessità  $O(V^m)$ , quindi esponenziale.  
Ora abbiamo una complessità lineare.

### LOSS FUNCTION

$$\mathcal{L}^{FF} = -\frac{1}{T} \sum_{i=1}^T \log P(w_i | w_{i-1} \dots w_{i-m+1})$$

## WORD EMBEDDINGS

Fin ora abbiamo rappresentato le parole come:

1) ONE HOT VECTOR:  $x \in \mathbb{Z}^{1 \times V}$  (o meglio  $x \in \{\emptyset, 1\}^{1 \times V}$ )

Se calcoliamo la cosine similarity fra due one-hot vector  $\cos(x_i, x_j)$ :

$$\cos(x_i, x_j) = \frac{\langle x_i, x_j \rangle}{\|x_i\| \cdot \|x_j\|}$$

Sarà sempre 0, perché  $x_i$  e  $x_j$  sono sempre ortogonali per definizione. Non ci serve a molto...

2) TF-IDF ZIKS:  $x \in \mathbb{R}^{1 \times V}$

Dove usiamo lo score di quella parola, inerente alla sua posizione. Questo è buono per fare query su dei documenti, non per le semplici parole che rimane problematico.

## WORD EMBEDDINGS

Un word embeddings  $x_i$  rappresenta la parola  $w_i$  con un vettore grande  $d \ll V$ .

$x \in \mathbb{R}^{1 \times d}$  dense representation

In questa rappresentazione densa non ci sono zeri (o quasi) la  $\cos(x_i, x_j) \neq 0$ .

Questa rappresentazione codifica in qualche modo il concetto di similitudine semantica, quindi più due parole hanno significati vicini più i loro embeddings sono "simili".

## IMPARARE I WORD EMBEDDINGS

Invece di usarli come parte di una NN, possiamo usare una NN specializzata a imparare i word embeddings.

Nel 2013 Nikolov et al presentarono una NN in grado di imparare i word embeddings. Questa NN è conosciuta come **Word2Vec**.

L'ipotesi che sta alla base di questa NN è la **distributional hypothesis** che recita:

"A word's meaning is given by the words that frequently appear close-by"

Vedremo due modelli per imparare i word embeddings:

- 1) CONTINUOUS BAG OF WORDS
- 2) SKIP GRAM

## CONTINUOUS BAG OF WORDS (CBOW) MODEL

Iniziamo con una parola detta **central word**  $w_c$ , e il suo  $2m$  contesto (**context words**):  $w_{c-m}, \dots, w_{c-1}, w_{c+1}, \dots, w_{c+m}$  (Le  $m$  parole a destra e  $m$  a sinistra di  $w_c$ ). Ogni parola  $w_i$  è associata a un one-hot embedding  $x_i$ .

In questo modello ci concentriamo nell'imparare a prevedere la  $w_c$  dato il suo contesto  $w_{c-m}, \dots, w_{c-1}, w_{c+1}, \dots, w_{c+m}$ :

$$P[w_c | w_{c-m}, \dots, w_{c-1}, w_{c+1}, \dots, w_{c+m}] = f(w_c)$$

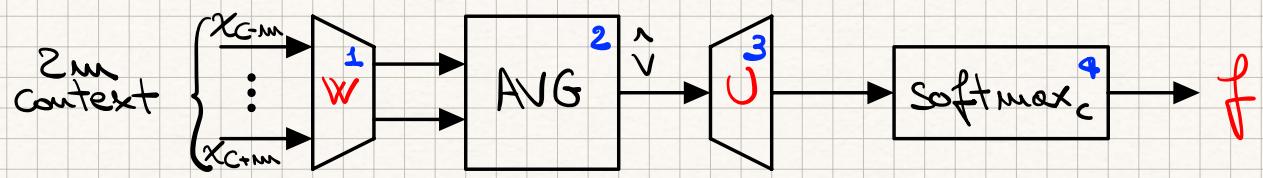
Impariamo la funzione  $f$  con:

- 1) Input embedding matrix  $W \in \mathbb{R}^{V \times d}$  dove le sue righe contengono i word embedding imparati.
- 2) Output embedding matrix  $U \in \mathbb{R}^{d \times V}$

## EQUAZIONI CBOW

- 1)  $v_i = x_i W \quad \forall i = c-m, \dots, c-1, c+1, \dots, c+m$
  - 2)  $\hat{v} = \frac{1}{2m} \sum_{\substack{i=c-m \\ i \neq c}}^{c+m} v_i$  : word embeddings medio del contesto
  - 3)  $\hat{z} = \hat{v} \cdot U$
  - 4)  $f(w_c) = \text{softmax } (\hat{z})$

## SCHÉMA



$$x_i \in \mathbb{R}^{1 \times v}$$

$$W \in \mathbb{R}^{v \times d}$$

$$v_i \in \mathbb{R}^{1 \times d}$$

$$\hat{v} \in \mathbb{R}^{1 \times d}$$

$$U \in \mathbb{R}^{d \times v}$$

$$f \in \mathbb{R}$$

## LOSS FUNCTION

Usiamo la cross-entropy loss

$$y_{CBOW} = -\log f(x_c) = -\log \frac{\exp(z_c)}{\sum_{i=1}^v \exp(z_i)} =$$

$$= -z_c + \log \sum_{i=1}^v \exp(z_i) = -\hat{v}_{x_c} + \log \sum_{i=1}^v \exp(\hat{v}_{x_i})$$

## SKIP GRAM (SG) MODEL

In questo modello facciamo il contrario del CBOW.

Possiamo delle  $w_c$  per arrivare al contesto.

$$P[w_{c-n}, \dots, w_{c-1}, w_{c+1}, \dots, w_{c+n} | w_c]$$

Questa probabilità è molto più complessa da calcolare. Abbiamo  $2n$  probabilità congiunte.

Per renderlo più semplice, facciamo delle assunzioni semplificative: **Naive Bayes Assumption**, che dice:

$$P[w_{c-n}, \dots, w_{c-1}, w_{c+1}, \dots, w_{c+n} | w_c] =$$

$$P[w_{c-n} | w_c] \dots P[w_{c-1} | w_c] \dots P[w_{c+n} | w_c]$$

Possiamo farlo solo se tutte le  $2n$  variabili sono **condizionalmente indipendenti** (Tutte dipendenti da  $w_c$ ).

Ora possiamo imparare con una NN:

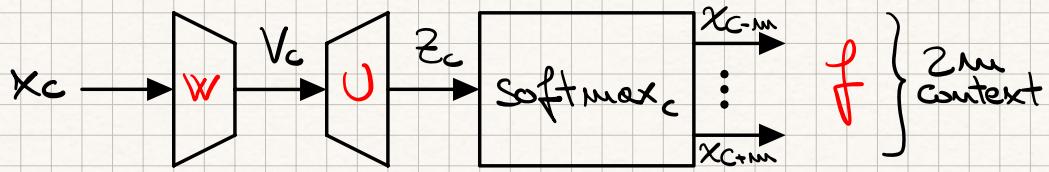
$$P[w_i | w_c] = f(w_c).$$

Come prima usiamo le stesse embedding matrix  $W$  e  $U$ .

**EQUAZIONI SG:**

- 1)  $v_c = x_c W$
- 2)  $z_c = v_c U$
- 3)  $f(w_c) = \text{softmax}(z_c) \quad \forall c = c-n, \dots, c-1, c+1, \dots, c+n$

## SCHÉMA



$$x_c \in \mathbb{R}^{1 \times V}$$

$$W \in \mathbb{R}^{V \times d}$$

$$v_c \in \mathbb{R}^{1 \times d}$$

$$U \in \mathbb{R}^{d \times V}$$

$$z_c \in \mathbb{R}^{1 \times V}$$

$$f \in \mathbb{R}$$

## LOSS FUNCTION

$$\mathcal{L}^{\text{SG}} = -\log \prod_{\substack{i=C-MN \\ i \neq c}}^{C+MN} f(x_i) = -\sum_{\substack{i=C-MN \\ i \neq c}}^{C+MN} \log \frac{\exp(V_c u_i)}{\sum_{j=1}^V \exp(V_c u_j)} =$$

$$= -\sum_{\substack{i=C-MN \\ i \neq c}}^{C+MN} V_c u_i + Z_MN \log \sum_{j=1}^V \exp(V_c u_j)$$