N-GRAM MODELS

Supponiamo di voler allenare un LM in grado di generare frasi in inglese composte da 4 lettere. Per esempio:

$$P[\text{We will rock you}] = P[\text{We}] \cdot P[\text{will} \mid \text{We}] \cdot P[\text{rock} \mid \text{We, will}] \cdot P[\text{you} \mid \text{We, will, rock}]$$

Per semplicità, assumiamo $N=2$. Da cui:

1) La probabilità di una parola dipende solo dalla parola che la precede, se presente
2) La probabilità di una parola, non dipende dalla posizione della parola nella frase.

Esempio:

1) $P[\text{rock} \mid \text{We, will}] = P[\text{rock} \mid \text{will}]$
2) $P[w_3 = \text{rock} \mid w_2 = \text{will}] = P[w_{i+1} = \text{rock} \mid w_i = \text{will}] \quad 1 < i < 3$

In generale la probabilità frase diventa:

$$P[\text{We}] \cdot P[\text{will} \mid \text{We}] \cdot P[\text{rock} \mid \text{will}] \cdot P[\text{you} \mid \text{rock}]$$

Estendiamo per $N>2$:

1) La probabilità di una parola dipende solo dalle $N-1$ parole che la precedono, se presenti
2) La probabilità di una parola, non dipende dalla posizione della parola nella frase.

FORMALISMO

Un n-gram model basato su V è un LM che assegna a $w_1 \dots w_k$ le seguenti probabilità

1) se $n=1$:

$$P[w_1 \dots w_k] = \prod_{i=1}^{k} P[w_i] \quad : \text{UNIGRAM}$$

2) se $n>1$:

$$P[w_1 \dots w_k] = P[w_1] \prod_{i=2}^{k} P[w_i \mid w_{i-1} \dots w_1]$$

che è circa uguale:

$$\approx P[w_1] \prod_{i=2}^{k} P[w_i \mid \underbrace{w_{i-1} \dots w_{i-n+1}}_{\text{le } n \text{ parole precedenti}}]$$

Stimiamo le probabilità dell'n-gram:

$$P[w_i] \approx \frac{\text{count}(w_i)}{\sum_{w \in V} \text{count}(w)} \rightsquigarrow tf$$

$$P[w_j \mid w_i \dots w_{j-1}] = \frac{\text{count}(w_i \dots w_{j-1} w_j)}{\sum_{w \in V} \text{count}(w_i \dots w_{j-1} w)}$$

Ovvero il numero di volte in cui è presente nel corpus la sequenza analizzata seguita da w_j diviso il numero di volte quella stessa sequenza segue qualsiasi parola (Inclusa w_j)

PAROLE NON PRESENTI NEL CORPUS

Per quanto riguarda parole usate a tempo di valutazione che non sono state viste a tempo di training?
The out of vocabulary words?
Dobbiamo introdurre un nuovo token speciale <UNK> per le parole sconosciute non presenti nel evaluation corpus. Bisogna convertire ogni parola non vista a tempo di training ma presente nell'evaluation corpus con <UNK>, e riusiamo l'evaluation come training per per stimare la probabilità del token <UNK>.

PROBLEMA DELLA SPARSITÀ

Il problema più grande degli n-gram model è la sparsità. Per evitarla si usa uno smoothing filter.
Il filtro più comune è il LAPLACE SMOOTHING (o anche detto add-one smoothing).
Questo filtro per il bi-gram si presenta così:

$$P[w_i \mid w_{i-1}] = \frac{count(w_i\ w_{i-1}) + 1}{\sum_{w \in V} count(w_{i-1}, w) + |V|}$$

È il filtro più facile.

LEARNING A BIGRAM

L'obiettivo, nel bigram, è calcolare:

$$P[w_i \mid w_{i-1}] = \frac{count(w_{i-1}, w_i)}{\sum_{w \in V} count(w_{i-1}, w)}$$

Per allenare il modello possiamo organizzare il tutto nella matrice dei pesi $W \in \mathbb{Z}^{V \times V}$ dove:

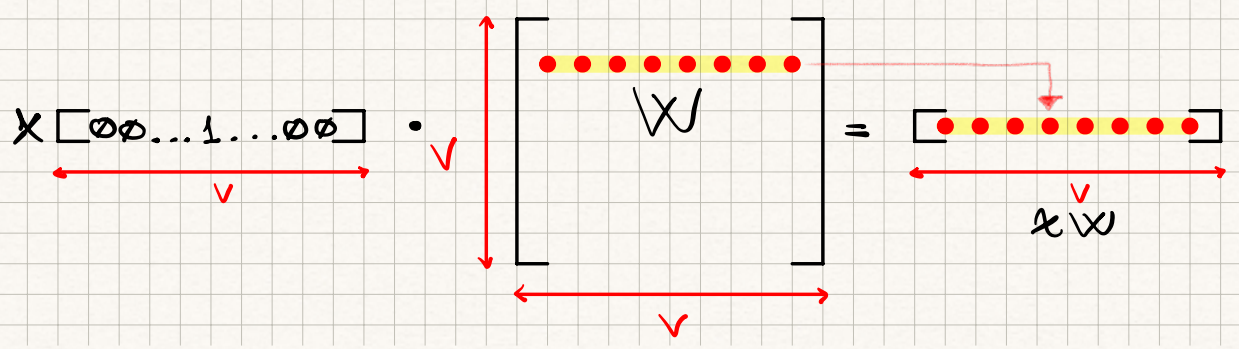
$$w_{ij} = count(w_i\ w_j)$$

ovvero il numero di volte in cui w_i segue w_j nel corpus. W è la matrice della co-occorrenze. Dove è facile intuire che:

$$\sum_{j=1}^{V} w_{ij} = count(w_i)$$

ONE-HOT VECTOR

Il modo più semplice per rappresentare matematicamente una parola è il one-hot vector $x \in \mathbb{Z}^{1 \times V}$ (Vettore riga) Che è un vettore pieno di zeri con un 1 a indicare la posizione in V della parola $w_i : x_i = 1$. Che poi indica la parola stessa in V.



x diffatto seleziona e' i-esima riga di W, corrispondente alla parola i: $x_i = 1$.

$$P[w_j | w_i] = \frac{Count(w_i w_j)}{Count(w_i)} = \frac{W_{ij}}{\sum_{j=1}^{V} W_{ij}} = \frac{W_{ij}}{W_j}$$

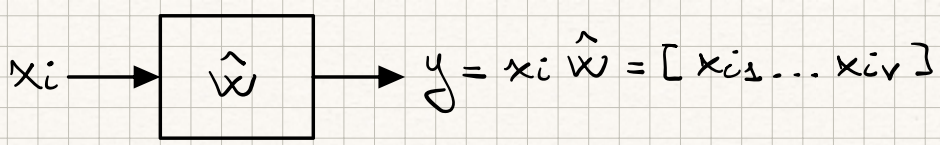
(w_j: Parola in V; W_{ij}: elemento matrice W)

LEARNING THE MATRIX W

Possiamo imparare la matrice W, inizializzandola, randomicamente.

La nostra rete neurale, prenderà una parola in ingresso w_i e produrrà in output la parola w_j (che reputa la più probabile per seguire w_i). Rappresentiamo w_i e w_j con i corrispondenti one-hot vector x_i e x_j.

La nostra NN ha un singolo livello lineare $\hat{W} \in \mathbb{R}^{V \times V}$ (Per semplicità incorporiamo il bias a $\hat{W}$):



Nota: dato che $\hat{W}$ contiene inizialmente cose a caso (positive ma anche negative) non possiamo direttamente associare y ai conteggi della parola come nella W ideale. Per tanto ne facciamo l'esponente per avere un qualcosa di simile che possiamo interpretare come conteggio:

$$y = x_i \hat{W} = [\hat{W}_{i1} \dots \hat{W}_{iV}] \rightarrow \text{Log-Count} \circ \text{Logit}$$

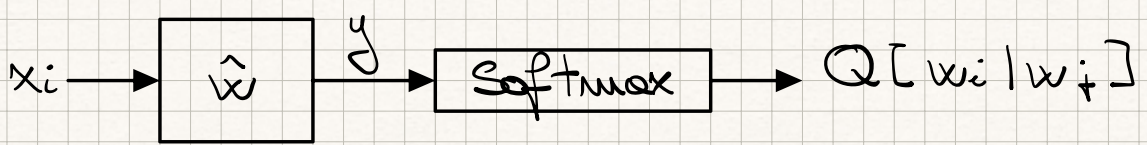
$$z = [e^{w_{i1}}, \ldots, e^{w_{iV}}] \rightarrow \text{"counts"}$$

Ora possiamo normalizzare le componenti di z per poter avere una distribuzione di probabilità di V. Possiamo assumere che questa distribuzione di probabilità sia il nostro LM approssimato Q:

$$Q[w_j | w_i] = \frac{\hat{e}^{w_{ij}}}{\sum_{j=1}^{V} \hat{e}^{w_{ij}}}$$

Abbiamo fatto tutta sta manata per riuscire ad avere pesi la cui somma è 1. Queste due operazioni di "esponenziare" e normalizzare vengono sempre insieme e le riassumiamo nella singola operazione di SOFTMAX:

$$y = x_i \hat{w}$$
$$Q[w_i | w_j] = \text{Softmax}(y)$$



La loss da usare è NLL:

$$\mathcal{L}^{bi} = -\frac{1}{T} \sum_{i=1}^{T} \log Q[w_i | w_j]$$