

Document Semantic Similarity

TIS Project

Alberto Pirovano Francesco Picciotti

Politecnico di Milano

5th May 2017



1 State of art

- NLP tradizionale
- Vector Space Model
- Deep Learning

2 Data Preparation

- Preprocessing
- Cleaning del testo

3 Word2Vec

4 Doc2Vec



Le tecniche adottate attualmente per trovare la **similitudine semantica tra testi** si basano su tre approcci:

- 1 NLP Tradizionale
- 2 Vector Space Model
- 3 Deep Learning based



- 1 State of art
 - NLP tradizionale
 - Vector Space Model
 - Deep Learning
- 2 Data Preparation
 - Preprocessing
 - Cleaning del testo
- 3 Word2Vec
- 4 Doc2Vec



Questo approccio si basa sull'utilizzo delle tradizionali tecniche di **Natural Language Processing** e si costituisce dei seguenti step:

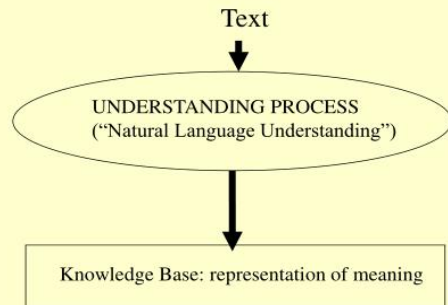
- Cleaning dei dati
- Pos-Tagging
- Stemming o Lemmatisation
- Parsing
- Ontologia

Tuttavia, dato che il nostro lavoro è molto **sensibile** e **dipendente** dalla qualità dei tool utilizzati, abbiamo trovato alcune consistenti **criticità** riguardanti:

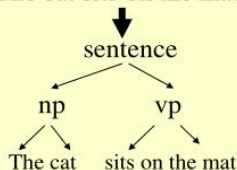
- **L'affidabilità** del Pos-Tagger italiano di TreeTagger
- **Reperire** una Ontologia e un parsing tool per la lingua italiana



NLP: the process



"The cat sits on the mat"



↓

Fact(type: statement,
agent: cat-002,
action: sits_on,
object: mat-001)

Fact(type: statement,
agent: Fido,
action: is_a,
object: cat)

Fact(type: statement,
agent: Freda,
action: loves,
object: Fido)

Natural Language Processing

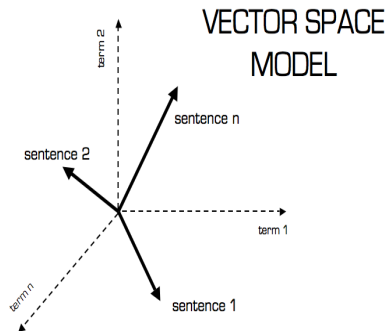


- 1 State of art
 - NLP tradizionale
 - **Vector Space Model**
 - Deep Learning
- 2 Data Preparation
 - Preprocessing
 - Cleaning del testo
- 3 Word2Vec
- 4 Doc2Vec



Vector Space Model - Explanation

Differentemente dal precedente, questo approccio ha le sue basi nello sviluppo di una **rappresentazione geometrica e vettoriale** per i documenti testuali. **Documenti e query** sono rappresentati da vettori con un numero di elementi pari al numero di termini presenti nel vocabolario. Tipicamente i termini sono le **parole distinte** presenti nell'insieme di documenti. A valle di questa rappresentazione vengono spesso utilizzate le **operazioni vettoriali** per confrontare due **documenti**.



Nel vector space model proposto da **Salton, Wong and Yang** i vettori sono composti da **weights**, ognuna associata ad un termine del dizionario e calcolata tramite **tf-idf**.

Considerando un **documento** d_j , questo viene rappresentato tramite un **vettore** d_j :

- $d_j = (w_{1,j}, w_{2,j}, \dots, w_{t,j})$ dove: $w_{i,j} = tf_{t,d} * \log \frac{|D|}{|\{d' \in D | t \in d'\}|}$



Latent Semantic Analysis è una tecnica di **Topic Modelling** che si colloca a valle del **document encoding** con **tfidf**.

È una tecnica di **feature extraction** che permette di migliorare significativamente la qualità di un lavoro di **clustering**.

Questa procedura viene usata per generare una categorizzazione di un **set di documenti** in un **set di topic** o anche per osservare le parole che descrivono un certo topic.

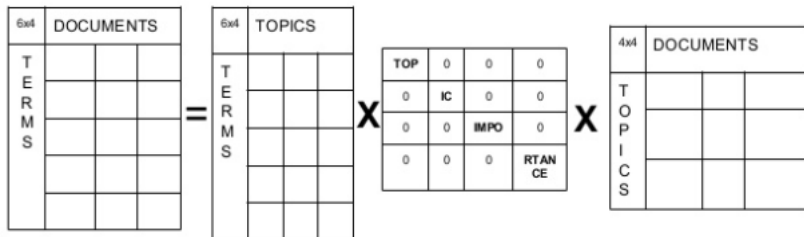
Si basa sulla creazione di una **Document-Term Matrix** nella quale le **righe** rappresentano le parole del **Bag Of Words** e ha una **colonna** per **documento** nel corpora.

Il cuore di questa procedura sta nella **riduzione della dimensionalità** di questa matrice tramite **SVD**.



Vector Space Model - LSA - SVD

Find three matrices U , Σ and V so that: $X = U\Sigma V^T$



Questa procedura ci permette di:

- **Estrarre** quanti topic desideriamo da un set di documenti.
- **Conoscere** la rilevanza di un certo topic dopo averlo estratto, in questo modo siamo in grado di fermare il processo di estrazione quando i topic cominciano a diventare poco significativi.
- **Categorizzare** documenti in topic
- **Descrivere** topics con le parole del **Bag Of Words**.



Vector Space Model - LSA - Example

In questo esempio possiamo osservare la riduzione di dimensionalità:

LSA is essentially low-rank *approximation* of document term-matrix

Word assignment to topics

IT cars

3	4	1	0
4	3	0	1
3	4	4	3
0	1	4	3
2	0	3	3
0	1	3	4

=

	IT	cars
linux	-0.33	-0.53
modem	-0.32	-0.54
the	-0.62	-0.10
clutch	-0.38	0.42
steering	-0.36	0.25
petrol	-0.37	0.42

X

Topic importance

11.4	
	6.27

X

Topic distribution across documents

	D1	D2	D3	D4
IT	-0.42	-0.48	-0.57	-0.51
cars	-0.56	-0.52	0.45	0.46

Il processo di LSA permette di costruire le 3 matrici che vediamo sopra, ognuna con una sua utilità:

- 1 Word assignment to topics
- 2 Topic importance
- 3 Topic distribution across documents

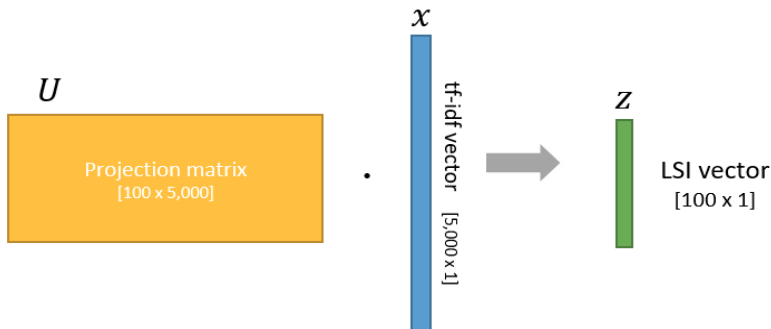


Vector Space Model - LSA - Example

Ipotizzando un numero di topics pari a 100 e la cardinalità del **Bag of Words** pari a 5000, per ottenere la rappresentazione dei documenti in termini di topics, cioè un vettore \mathbf{z} , dobbiamo:

- 1 **Vettorizzare** un documento in \mathbf{x} .
- 2 **Proiettare** il **tfidf** vector \mathbf{x} sul topic space.

Se definiamo la matrice $U = (\text{Wordassignment to topics})^T$ possiamo visualizzare la **proiezione** in questo modo:



Vector Space Model - Algorithm

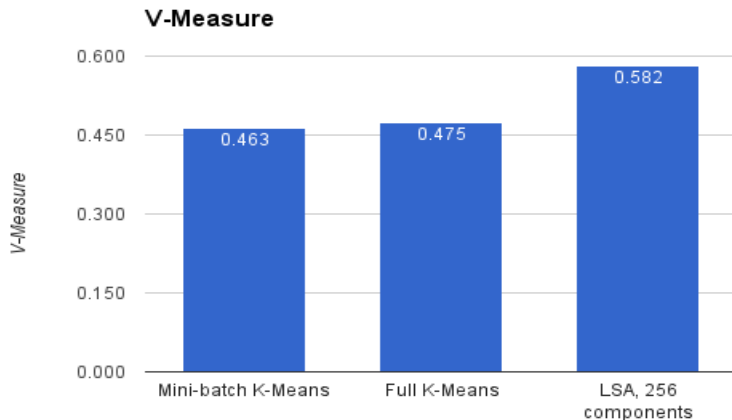
Riassumendo, possiamo individuare i seguenti **passaggi**:

- 1 Cleaning dei dati (Stemming o Lemmatisation)
- 2 Vectorization using TF/IDF (\mathbf{x} nella figura nella slide precedente)
- 3 LSA (optional) (\mathbf{z} nella figura precedente)
- 4 Clustering using similarity measure (Cosine, Pearson, ...) tra \mathbf{x} vectors o \mathbf{z} vectors



Vector Space Model - LSA - Performance

Come possiamo vedere nella figura, utilizzare le classiche tecniche di **clustering** senza **LSA** può portare una riduzione rilevante delle **performances**:



Vector Space Model (con LSA) - Pros and Cons

Cons:

- Non adatto a trattare **lunghi documenti**, infatti a causa della alta **dimensionalità** il valore **tfidf** dei componenti si riduce, riducendo di conseguenza il **dot-product**. (LSA fixes)
- **Fortemente sensibile a Falsi Negativi**, infatti documenti nello stesso **contesto** ma con diversa terminologia non saranno considerati simili. (LSA fixes)
- **Perde l'ordine delle parole.**
- **Pre-processing** dipendente.
- **Le dimensioni generate possono essere difficili da interpretare**, sensate matematicamente ma non dal punto di vista del natural language



Pros:

- Modello semplice basato sull'**algebra**.
- Le **weights** non sono binarie.
- Permette di calcolare un grado di **similarità continuo**.

- 1 State of art
 - NLP tradizionale
 - Vector Space Model
 - Deep Learning
- 2 Data Preparation
 - Preprocessing
 - Cleaning del testo
- 3 Word2Vec
- 4 Doc2Vec



Il terzo approccio che proponiamo è molto diverso dai precedenti, infatti si basa sull'utilizzo di **neural networks**. La fondamentale differenza è che, mentre il secondo **definiva un algoritmo** per determinare le rappresentazioni vettoriali delle parole (tfidf), questo **definisce una neural network** con la task di imparare quell'algoritmo dai dati. Questa **NN** può essere **costruita** in due diversi modi, ognuno dei quali descrive **come** imparare la **word-representation** per ogni parola:

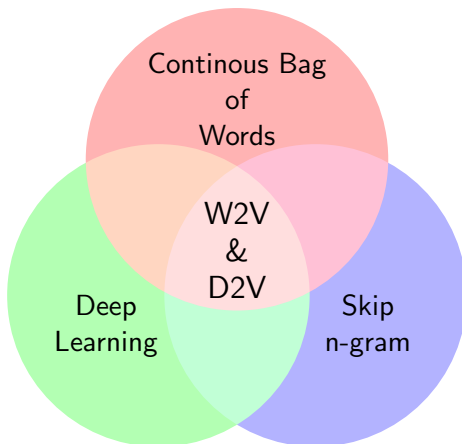
- **Continuous Bag Of Words model**
- **Skip Grammar model**

Dato che il processo di apprendimento è **unsupervised**, questi modelli permettono di **determinare la task** tramite la definizione di un target per ogni input.



Deep Learning based

Questa tecnica quindi si basa su 3 ambiti:



Deep Learning based: Word2Vec & Doc2Vec

Questo metodo è stato implementato da **Google** sotto il nome di **Word2Vec** e **Doc2Vec**. Il primo permette di determinare le **relazioni semantiche** che un particolare **corpus di testi** assegna ad un **Bag Of Words** di parole. **Doc2Vec** invece è una tecnica che si configura come una **estensione di Word2Vec** la quale, preso in ingresso un set di documenti (corpora), genera un **grado di similarità**.



Deep Learning based - Pros and Cons

Dopo una ampia **discussione**, seguita da una approfondita **analisi critica** di questi due approcci, siamo giunti alle seguenti **conclusioni**, che in termini di pro e contro si possono riassumere nel seguente modo:

Pros:

- Molto meno dipendente da un preprocessing
- Combina il metodo *Geometrico* con quello *NLP Tradizionale*
- **Non sfrutta una ontologia, ma la crea**
- **Language independent**
- **Tiene in considerazione anche l'ordine delle parole, Context-aware**

Cons:

- Tecnica **unsupervised**
- Necessità di un esperto per **validare** la similarità
- Può risultare in **GIGO** system (Garbage In Garbage Out)



- 1 State of art
 - NLP tradizionale
 - Vector Space Model
 - Deep Learning
- 2 Data Preparation
 - Preprocessing
 - Cleaning del testo
- 3 Word2Vec
- 4 Doc2Vec



"Preprocessing is 80% of NLP work"

Lev Konstantinovskiy

Il **dataset** si suddivide in due corpora:

- il corpus del **Sole 24 Ore** con 3265 articoli, di cui 31 non hanno body
- il corpus di **Radiocor** con 6916 articoli

Il corpus prima del **preprocessing** contiene quindi 10150 articoli.

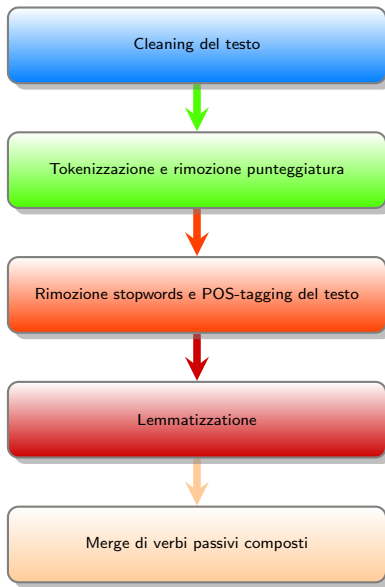
Togliendo i **duplicati** otteniamo 9283 articoli, cioè ci sono 867 articoli duplicati.



- 1 State of art
 - NLP tradizionale
 - Vector Space Model
 - Deep Learning
- 2 Data Preparation
 - Preprocessing
 - Cleaning del testo
- 3 Word2Vec
- 4 Doc2Vec



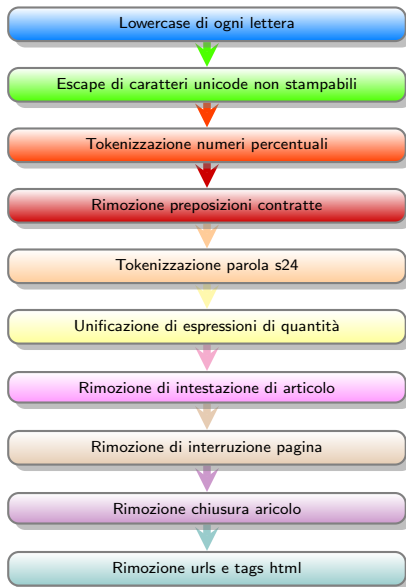
Pipeline completa



- 1 State of art
 - NLP tradizionale
 - Vector Space Model
 - Deep Learning
- 2 Data Preparation
 - Preprocessing
 - Cleaning del testo
- 3 Word2Vec
- 4 Doc2Vec



Cleaning pipeline



- 1 State of art
 - NLP tradizionale
 - Vector Space Model
 - Deep Learning
- 2 Data Preparation
 - Preprocessing
 - Cleaning del testo
- 3 Word2Vec
- 4 Doc2Vec



Word2Vec è un gruppo di modelli che vengono usati per generare **word-embeddings** da **unstructured text**. Per questo scopo viene usata una **two-layers neural network** che viene trainata per ricostruire il **contesto linguistico** delle parole.

- **Input: corpus** di testo
- **Output: vector-space**

Ogni parola **unica nel corpus** verrà rappresentata con un vettore nel **vector-space**. L'algoritmo genererà degli word-embeddings tali per cui parole che condividono lo stesso contesto nel corpus risulteranno vicine nel **vector space**.



Word2Vec - From unsupervised to supervised

Questa task può essere definita in due modi:

- ① **Skip Grammar** - predire il **contesto** data la parola in input
 - **Input:** focus-word
 - **Target:** context-words
- ② **Continuous Bag Of Words** - predire la parola in input dato il **contesto**
 - **Input:** context-words
 - **Target:** focus-word

Definendo input e target pairs abbiamo definito un **supervised problem**, che affronteremo usando **logistic classifiers** (logistic regression).

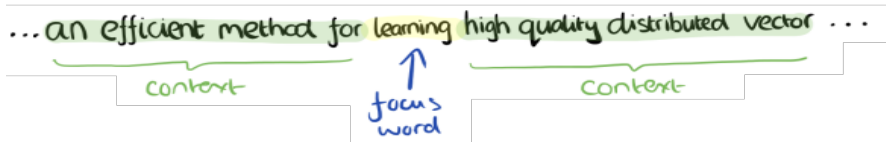


Gli **input-target pairs** sono rappresentati tramite **One Hot Encoding**, mentre l'**output** della NN sarà un vettore di probabilità di dimensione pari al numero di parole del **vocabolario**.

Di conseguenza se abbiamo un vocabolario di **dimensionalità V** avremo dei **vectors** costituiti da V elementi di cui un 1 e gli altri 0.



In entrambi i casi è necessario definire un **context** per la parola in input, che identifichiamo con un parametro chiamato $c = \text{window size}$. Tramite questa *window size*, una volta fissata una **input word/focus word**, chiamiamo **context** l'insieme delle c parole prima della input word più le c parole dopo la input word.

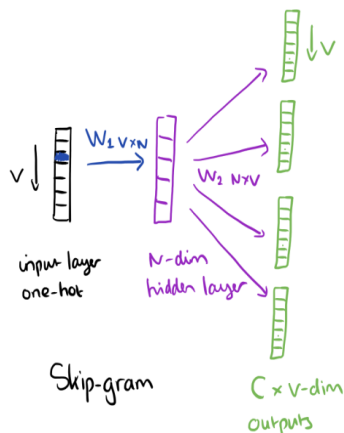


Terminologia:

- V = numero di elementi nel vocabolario
- N = numero di weights per word embeddings
- c = window size



Word2Vec - Skip Grammar



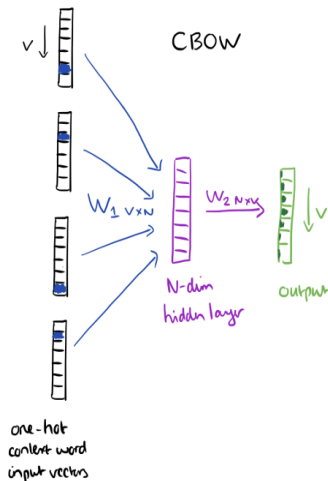
Data una sequenza di **training words** w_1, w_2, \dots, w_T , con $T = \text{trainingsetsize}$, il **training objective** di questo modello è trovare il parametro θ che massimizza la **Log-Likelihood**:

$$J_{\theta} = \frac{1}{T} \cdot \sum_{t=1}^T \sum_{-c < j < c, j \neq 0} \log p(w_{t+j} | w_t)$$

Ogni classification task, data una **focus word** w_t e una parola del contesto w_{t+j} , calcola la **probabilità** $p(w_{t+j} | w_t)$.

- Di conseguenza per ogni input vector definiamo circa **2c logistic classifiers**.





Data una sequenza di **training words** w_1, w_2, \dots, w_T , con $T = \text{trainingsetsize}$, il **training objective** di questo modello è trovare il parametro θ che massimizza la **Log-Likelihood**:

$$J_{\theta} = \frac{1}{T} \cdot \sum_{t=1}^T \log p(w_t | w_{t-n}, \dots, w_{t+n})$$

Ogni classification task, date una serie di **context words**, le calcola la **probabilità** $p(w_t | w_{t-c}, \dots, w_{t+c})$.

- Di conseguenza per ogni input vector definiamo **1 logistic classifier**.



Le due **weight matrices** $W_1[V \cdot N]$ e $W_2[N \cdot V]$ sono della stessa dimensione e contengono entrambe un **word-embedding vector** per ogni vocabulary word.

La **prima** fornisce la **input vector representation** delle vocabulary words e la **seconda** fornisce la **output vector representation** delle vocabulary words.

In generale data una parola $w \in V$:

- v_w è la sua **input representation**, vale a dire un vettore $[1 \cdot N]$ relativo a w che otteniamo da W_1 :
- v'_w è la sua **output representation**, vale a dire un vettore $[1 \cdot N]$ relativo a w che otteniamo da W_2 :



Word2Vec - Hidden layer

È interessante notare come, dato un input **OHE**, l'**hidden layer** funga da **lookup table** per selezionare il **word vector** relativo alla input word, v_{wInput} (no activation function).

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = \begin{bmatrix} 10 & 12 & 19 \end{bmatrix}$$

Possiamo quindi dire che:

- **Skip Grammar:** L'**output dell'hidden layer** (v_{wInput}) è il **word vector** relativo all'**input word**, in questo caso $v_{wInput} = [10, 12, 19]$.
- **CBOW:** Ogni **OHE** vector delle parole del contesto effettua il lookup e ottiene il word-embedding dalla matrice W_1 . Successivamente viene fatta una media degli word embeddings ottenuti in modo tale da ottenere l'output dell'hidden layer ($v_{wAverage}$).



Considerando i due tipi di approccio, possiamo quindi esplicitare le **probabilità scalari** utilizzate nella funzione di costo:

- **Skip Grammar:** $p(w_{t+j}|w_t) = \frac{\exp(v_{wInput}^T \cdot v'_{w_{t+j}})}{\sum_{i=1}^V \exp(v_{wInput}^T \cdot v'_{w_i})}$
- **CBOW:** $p(w_t|w_{t-c}, \dots, w_{t+c}) = \frac{\exp(v_{wAverage}^T \cdot v'_{w_t})}{\sum_{i=1}^V \exp(v_{wAverage}^T \cdot v'_{w_i})}$

Tecnicamente il **vettore di probabilità** che viene generato dalla rete, dato un input vector o un input average, è:

- **Skip Grammar:** $y_{t+j} = \frac{\exp(v_{wInput}^T \cdot W_2)}{\sum_{i=1}^V \exp(v_{wInput}^T \cdot v'_{w_i})}$
- **CBOW:** $y_t = \frac{\exp(v_{wAverage}^T \cdot W_2)}{\sum_{i=1}^V \exp(v_{wAverage}^T \cdot v'_{w_i})}$



Word2Vec - Output layer

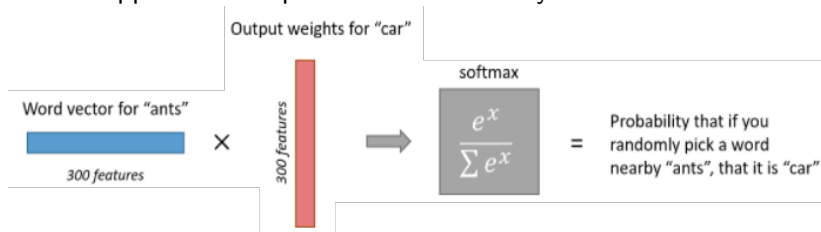
L'**output layer** si compone di due operazioni:

- 1 Calcolare lo score dell'input vector, $scoreVec = v_{wInput}^T \cdot W_2$ per **Skip Grammar** o $scoreVec = v_{wAverage}^T \cdot W_2$ per **CBOW**.

- 2 **Multi-class classification** attraverso **softmax**,

$$softmax(scoreVec) = \frac{e^{scoreVec}}{\sum_{i=1}^V e^{scoreVec_i}}$$

Dove $V = sizevocabulary$, $scoreVec_i$ è l'i-esimo componente di $scoreVec$ e e^{vector} rappresenta l'esponenziale element by element.



L'aspetto interessante di questa tecnica è che, dopo il training, **la rete non verrà utilizzata per la task su cui è stata addestrata**. Il training serve solo per aggiornare e migliorare i pesi delle matrici W_1 e W_2 .

- Quello che ci interessa ottenere dopo la fase di training sono le weights della matrice W_1 che rappresenteranno gli **word vectors** per le parole del vocabolario (row-wise).



Il modello presentato ha dei problemi nel momento in cui si vuole applicare la backpropagation per l'aggiornamento delle weights. Infatti abbiamo due matrici da aggiornare che hanno in media 1M di weights ciascuna. Questo risulta in alcune problematicità, come ad esempio:

- time complexity.
- abbiamo bisogno di molti training data per evitare over-fitting.



Nel secondo paper gli autori di questa tecnica hanno proposto 3 miglioramenti:

- Phrases identification.
- Subsampling common words.
- Modificare il training objective usando la tecnica del negative sampling che porta ogni training example ad aggiornare solo una piccola percentuale di weights.

Questo procedimento ha anche significativamente migliorato le performance del modello.



Ci soffermiamo sulla terza tecnica perchè è la più interessante.

- W_1 : Aggiorno solo il **weight vector** relativo alla input word, ma questo avviene a prescindere dal "**negative subsampling**".
- W_2 : Invece di aggiornare tutti gli **weight vectors** della matrice W_2 ad ogni training sample processato, prevede di selezionare un sottoinsieme di "negative words" da aggiornare.

In questo contesto "negative word" corrisponde ad una parola per cui il target vector presenta uno 0.

Solitamente il numero di negative selections che vengono fatte sono da 5 a 20.



Per selezionare gli **word vectors della matrice** W_2 da aggiornare consideriamo solo quelli riferiti alle parole più frequenti nel testo. La probabilità per una parola di essere campionata è:

- $$p(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=0}^V f(w_j)^{\frac{3}{4}}}$$

In questo modo possiamo **ridurre drasticamente il numero di weights che vengono aggiornate ad ogni training sample**, indicativamente con 3M di weights e 1800 aggiornamenti con "negative sampling" aggiorniamo lo 0.06% delle weights.



Se due parole vengono usate in contesti simili, allora il modello deve generare risultati molto simili per queste due parole.

Un modo che la NN ha per fare predizioni simili per queste due parole è utilizzare due **word vectors** simili.

La rete così risulta motivata a imparare simili **word vectors** per queste due parole.



- 1 State of art
 - NLP tradizionale
 - Vector Space Model
 - Deep Learning
- 2 Data Preparation
 - Preprocessing
 - Cleaning del testo
- 3 Word2Vec
- 4 Doc2Vec



Qui spaghiamo per bene come funziona doc2vec

