L’algebra del significato

Paolo Caressa

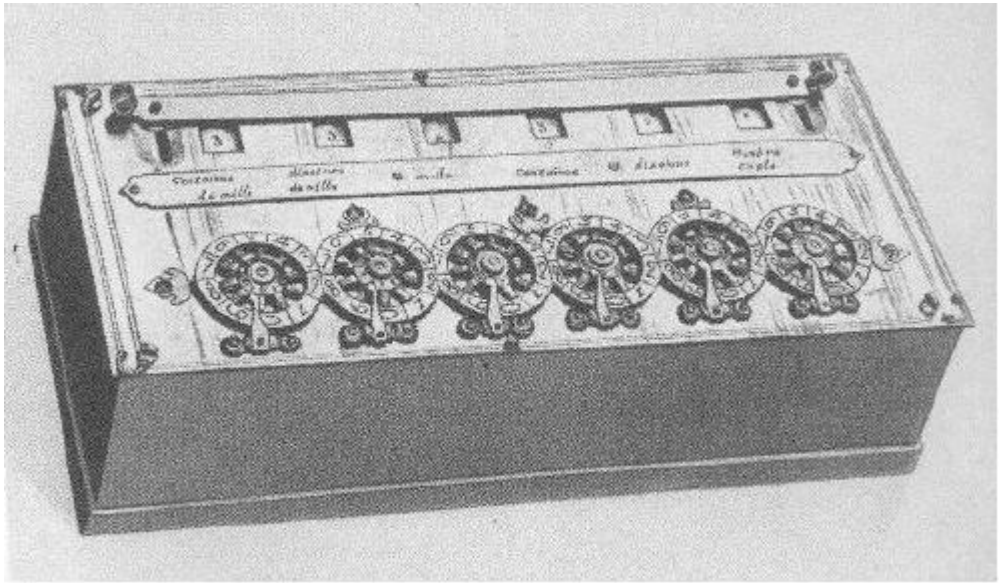
Latina, 12 marzo 2021

https://www.linkedin.com/in/paolocaressa
https://github.com/pcaressa
@www_caressa_it
http://www.caressa.it

Agenda

1. Il sogno di Leibniz
2. Partire dalla sintassi
3. Partire dal testo
4. Punti, vettori e parole
5. Digressione: le reti neurali ricorrenti
6. Algebra lineare di parole e concetti
7. Capire il contesto

1. Il sogno di Leibniz

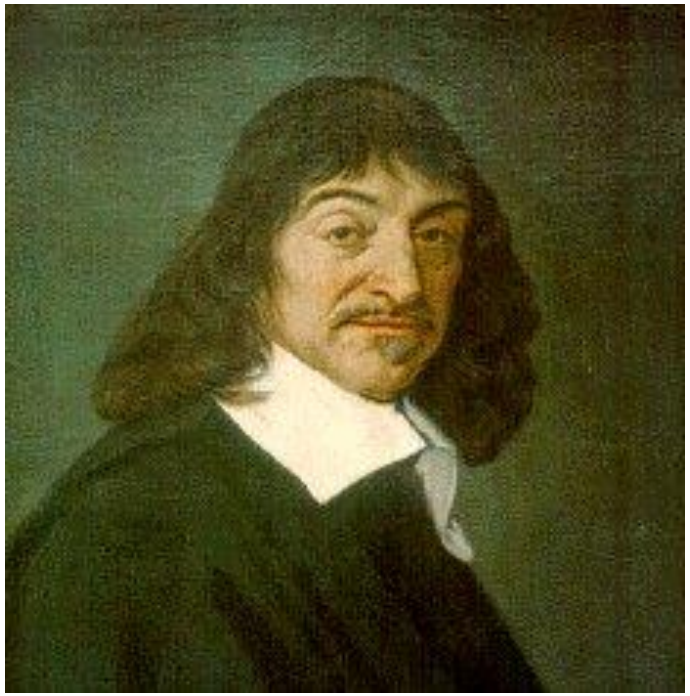


Signori: calcoliamo!

«L'unico modo di rendere corretti i nostri ragionamenti è di farli tangibili come quelli dei matematici, in modo da trovare gli errori semplicemente guardando, e quando ci siano dispute fra persone si possa semplicemente dire: calcoliamo senza indugio per vedere chi ha ragione.»

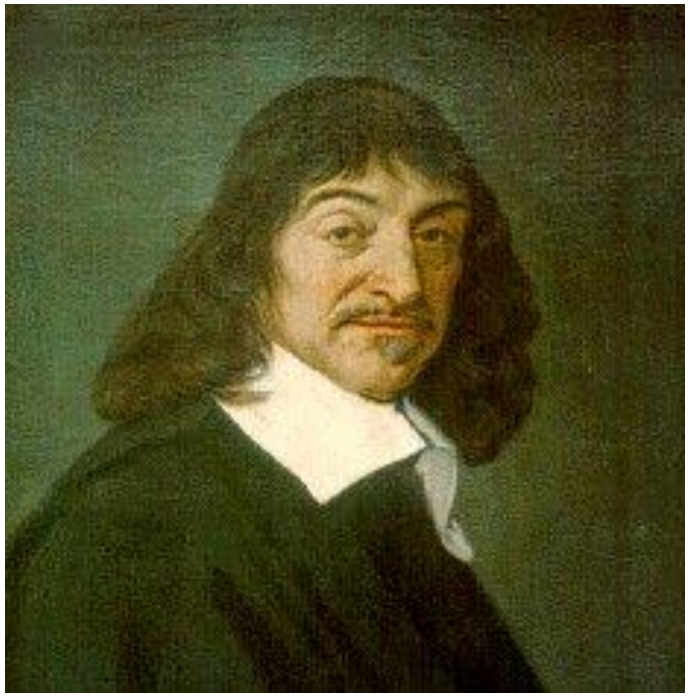
Arte della scoperta (1685)

… pensando a Cartesio



Il grande filosofo francese aveva nel XVII secolo «algebrizzato» la geometria, consentendo di sostituire ragionamenti e costruzioni geometriche con calcoli algebrici: Leibniz voleva fare lo stesso per qualsiasi tipo di ragionamento e deduzione, esprimibili nel linguaggio umano...

Un sogno che prende forma...



Le tecniche di w*ord embedding* realizzano in qualche modo il sogno di Leibniz usando proprio i metodi di Cartesio!!!

2. Partire dalla sintassi

Natural Language Processing

La moderna *teoria dei linguaggi formali*, che si potrebbe chiamare anche «linguistica matematica» (nota anche come *linguistica computazionale*), tenta di formalizzare matematicamente il linguaggio e renderlo «processabile» in modo automatico (da un computer): siamo nel solco logico-combinatorico caro a Leibniz.

Lo studio della sintassi e il suo legame con la teoria degli automi (*teoria generativa* di Noam Chomsky) ha dato grandi frutti con le lingue artificiali: i linguaggi di programmazione.

La triplice natura del linguaggio

Ma le lingue umane non sono semplici e precise come quelle artificiali! Le lingue umane sono caratterizzate almeno da (C.W. Morris, 1938):

- *Sintassi*: la struttura delle sequenze di simboli che compongono parole, frasi, etc. (analisi grammaticale).
- *Semantica*: i significati possibili di un testo (analisi logica).
- *Pragmatica*: il contesto (= con testo) di riferimento del significato (comprensione del testo).

Il significato del testo è l'oggetto centrale che si proietta in ciascuna di queste tre dimensioni concettuali.

Un esempio classico

Il *sillogismo categorico,* codificato da Aristotele, è una formalizzazione *sintattica* di un frammento del linguaggio per svolgere deduzioni in modo automatico: l'esempio standard è

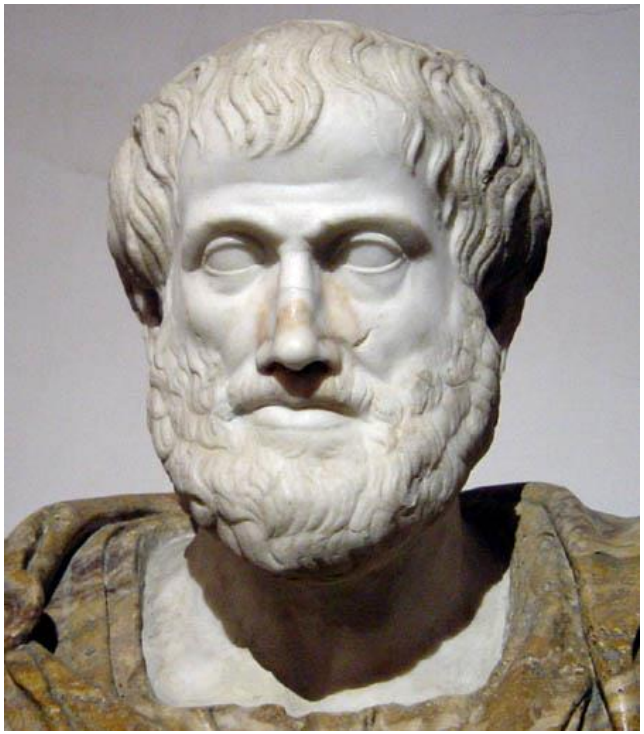
Socrate e Platone sono uomini

Gli uomini sono mortali

dunque

Socrate e Platone sono mortali

Indubitabile, no?



L’ambiguità della lingua

O no? Guardate cosa si è inventato il grande matematico e logico Giuseppe Peano:

Pietro e Giovanni sono apostoli

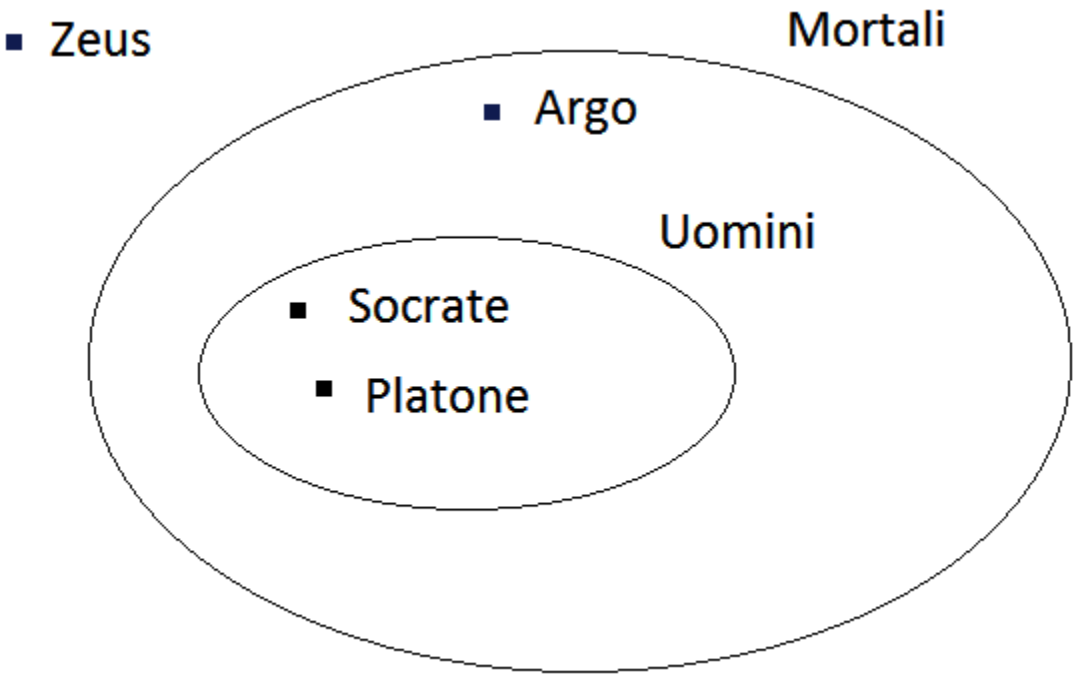
Gli apostoli sono dodici

dunque

Pietro e Giovanni sono dodici

Dare una semantica

Una *interpretazione* di un linguaggio è un modo di "mappare" la sintassi di linguaggio sulla sua semantica, cioè su oggetti di un mondo possibile denotati da entità sintattiche.



Formalizzazione del sillogismo

$Socrate \in Uomini \wedge Platone \in Uomini$,

$Uomini \subset Mortali$

$\vdash Socrate \in Mortali \wedge Platone \in Mortali$

Infatti se A e B sono insiemi, allora $A \subset$ B se e solo se per ogni $x \in A$ si ha pure che $x \in B$.

Che è una deduzione corretta: cioè la validità semantica del sillogismo è garantita da un teorema di teoria degli insiemi, cioè sillogismo = sintassi, teoria degli insiemi = semantica.

Formalizzazione del paradosso

Nel caso dei «dodici apostoli», il dire *gli apostoli sono dodici* è una abbreviazione per il *numero di elementi dell'insieme degli apostoli è 12*: dunque una possibile formalizzazione è

$Pietro \in Apostoli \wedge Giovanni \in Apostoli,$

$\#Apostoli = 12$

$\vdash \#\{Pietro, Giovanni\} = 12$

Ora è ovvio che si tratti di una falsa deduzione perché non è un teorema di teoria degli insiemi!

Separare sintassi e semantica!

Hanno dunque ragione i linguisti! Per sciogliere l'ambiguità linguistica occorre separare fra sintassi e semantica individuando i domini semantici di ciascuna frase.

Questo semplice esempio mostra come si possa studiare (parte di) un linguaggio usando i sistemi formali della logica matematica per la sintassi e la teoria degli insiemi, o dei suoi frammenti, per la semantica.

Ma per capire il contesto tutto ciò non basta…

Il ruolo del contesto...

«Ho visto mangiare un cane»

Il ruolo del contesto...

«Ho visto mangiare un cane»

3. Partire dal testo

Mettiamo ordine fra le parole

Una prima idea per far emergere la *semantica* di un testo dalla sua struttura è notare che le parole in una frase sono disposte in un certo *ordine*, dettato dalle regole grammaticali e da ciò che si vuole esprimere.

```
Paolo cuoce la pasta e poi la mangia
Paolo mangia la pasta e poi la cuoce
Poi la pasta la cuoce Paolo e mangia
La pasta la mangia e poi cuoce Paolo
Cuoce e la la mangia Paolo pasta poi
```

La grammatica non basta

In un testo, dopo una parola non ne può seguire una qualsiasi altra, ma solo quelle che *ha senso* porre dopo la parola data, sia in termini sintassi che semantici.

Per esempio, in italiano prima del verbo va solitamente un soggetto. Tuttavia la frase

```
Il gatto abbaia
```

è grammaticalmente corretta ma priva di senso: il verbo segue il soggetto, ma non è il verbo giusto associato al soggetto giusto...

Andrej Andreevič Markov

La prima applicazione (1913) delle sue «catene di Markov» è stata il conteggio delle occorrenze di lettere nell'*Eugeny Onegin* di Puškin, per capire quando una vocale segue un'altra vocale (12,8% dei casi) o una consonante (66,3% dei casi).

Un modello markoviano

Si può applicare l'idea di Markov non alle singole lettere, ma alle parole. Denotiamo con P_{xy} la probabilità che la parola x segua la parola y. Per esempio, se $P_{xy} = 0$ vuol dire che la parola x non compare mai dopo la parola y.

Avendo a disposizione un corpus di testi in una lingua, possiamo stimare P:

$$P_{xy} = \frac{numero\ di\ volte\ che\ x\ segue\ y}{numero\ di\ volte\ che\ x\ segue\ una\ qualsiasi\ parola}$$

Si tratta di un modo «frequentista» di stimare le probabilità.

Generare catene di parole…

Se per tutti i documenti di un corpus calcoliamo P_{xy} per tutte le coppie di parole che occorrono una di seguito all'altra, fissata y le P_{xy} danno una distribuzione di probabilità che possiamo usare per estrarre a caso una parola x fra quelle che *possono* seguire y.

Poi estraiamo una parola w fra quelle che possono seguire x, e così via, in modo da generare sequenze di parole che sono possibili nel linguaggio dato in base all'uso che se ne fa nei testi disponibili.

Usiamo questa tecnica per un esperimento un po' surreale...

Un semplice esperimento markoviano

Ecco un semplicissimo programma *che risponde a domande come avrebbe fatto Antonio Gramsci*: per scriverlo ho

- scaricato da Internet i tomi degli "Scritti politici" di Gramsci https://www.liberliber.it/online/autori/autori-g/antonio-gramsci/
- scritto una libreria per calcolare le probabilità P_{xy} per questo corpus (la parte più divertente del *coding*).
- scritto un programma che, usando la libreria, chiede di inserire una domanda e, partendo dall'ultima parola di quest'ultima, generi una risposta (5 parole per ogni parola data, per consentire frasi parzialmente sensate).

Un semplice esperimento markoviano

Ovviamente il risultato non ha molto senso: si chiama...

Conversazione con un pazzo che si crede Gramsci

Potete scaricare il programma Python e i dati necessari da questa URL: **https://github.com/pcaressa/gramsci**

(cambiando il contenuto del file "gramsci.txt", cancellando il file .pkl e rilanciando il programma potrete farlo conversare alla maniera di chi volete voi...)

Questi programmi si chiamano *bullshit generator* (sic).

Probabilità condizionate

I valori P_{xy} che forniscono le probabilità che una parola x segua una parola y si possono pensare anche come le probabilità che la parola x appaia nel testo, avendo l'informazione che la parola y è appena apparsa: sono quindi quelle che i probabilisti chiamano *probabilità condizionate* P(x|y).

Per queste probabilità esiste una tecnica deduttiva fondamentale in moltissimi campi applicativi che possiamo usare per studiare la struttura statistica di un linguaggio sulla base di testi scritti per mezzo di esso: *l'inferenza bayesiana*, che si basa sul *teorema di Bayes*.

Thomas Bayes (1702-1761)

Modelli bayesiani

Ricordiamo brevemente in cosa consista il *teorema di Bayes*: supponiamo di avere un insieme di ipotesi h_1, ..., h_2 relative a un insieme di dati (o osservazioni) d.

Vogliamo scegliere l'ipotesi più probabile dato d, quindi scegliere l'h_i per cui è massima la probabilità $P(h_i|d)$.

Il problema è che tipicamente questa «probabilità a posteriori» spesso non è misurabile: si pensi all'esempio di una malattia (h) da dedurre in base ai sintomi (d); di solito sappiamo stimare la probabilità di avere certi sintomi data la presenza della malattia, cioè $P(d|h_i)$.

Il teorema di Bayes

Il teorema di Bayes consente di calcolare la probabilità condizionata di h_i dato d a partire dalla probabilità condizionata di d dato h_i e dalle loro probabilità «a priori»:

$$\mathrm{P}(h_i|d) = \frac{\mathrm{P}(d|h_i)\mathrm{P}(h_i)}{\mathrm{P}(d)}$$

Non sorprende che $\mathrm{P}(h_i|d)$ sia proporzionale sia a $\mathrm{P}(h_i)$ sia a $\mathrm{P}(d|h_i)$. Inoltre, più è elevata la probabilità a priori $\mathrm{P}(d)$, più è facile che d sia osservata indipendentemente da h_i, e qiundi più decresce la probabilità $\mathrm{P}(h_i|d)$.

Scegliere l'ipotesi migliore

A questo punto, possiamo calcolare le probabilità condizionata per ciascuna ipotesi e scegliere quella che le massimizza:

$$\begin{aligned} h_{best} &= \arg\max_i \mathrm{P}(h_i|d) \\ &= \arg\max_i \mathrm{P}(d|h_i)\mathrm{P}(h_i)/\mathrm{P}(d) \\ &= \arg\max_i \mathrm{P}(d|h_i)\mathrm{P}(h_i) \end{aligned}$$

Se le h_i sono equiprobabili abbiamo la *Maximum Likelihood* (in quanto possiamo non considerare P(h_i) nel calcolare l'indice *i* per il quale l'argomento è massimo)..

Classificazione bayesiana di documenti

- Consideriamo un corpus di documenti dai quali estraiamo tutte le parole e le poniamo in un «vocabolario» V.
- Un singolo documento d viene rappresentato in questo modo: se n è il numero delle sue parole (anche ripetute), il documento si identifica con una funzione $d:\{1,2,...,n\}\rightarrow V$ in modo che associ al valore i la parola in quella posizione. In altri termini il documento si rappresenta con la successione delle parole che lo compongono nell'ordine in cui si presentano.
- Un documento possiede uno ed un solo tag: diciamo che il documento d appartiene alla *classe* h se il suo tag è h.

Ipotesi «naive Bayes»

Nell'applicare l'algoritmo di Bayes

$$h_{best} = \arg\max_i \mathrm{P}(h_i|d)$$

d è nel nostro caso una funzione, determinata dall'insieme {*d*(1),...,*d*(n)} dei suoi valori. A priori non è detto che questi valori siano indipendenti (per esempio a un articolo seguirà un nome): l'ipotesi "ingenua" è che lo siano e quindi

$$h_{best} = \arg\max_i \left[\mathrm{P}(d(1)|h_i) \cdots \mathrm{P}(d(n)|h_i)\right]\mathrm{P}(h_i)$$

(*a* e *b* sono indipendenti se P(*a*|*b*) = P(*a*) e P(*b*|*a*) = P(*b*), da cui segue che P(*a*,*b*) = P(*a*)P(*b*).)

Ipotesi «naive Bayes»

L'ipotesi «ingenua» è quindi in questo caso che le posizioni delle parole non influenzino le loro probabilità di comparire in una data posizione.

Questo è chiaramente falso ed è la negazione dell'ipotesi alla base del modello markoviano!

Tuttavia questa palese falsità semplifica computazionalmente l'algoritmo in modo formidabile e quindi in prima approssimazione la diamo per buona...

Ipotesi «naive Bayes»

Nel caso specifico dei documenti di testo, questa ipotesi si può formalizzare come, per ogni *h*, *k*

$$\mathrm{P}(d(h) = w|h_i) = \mathrm{P}(d(k) = w|h_i)$$

Cioè, data una classe, le parole di un documento hanno la stessa probabilità a posteriori di comparire indipendentemente dalla posizione che occupano nel documento.

Ipotesi «naive Bayes»

In questo modo possiamo stimare le probabilità $\mathrm{P}(d(h) = w|h_i)$ semplicemente come $\mathrm{P}(w|h_i)$, cioè calcolando la frequenza di *w* in tutti i documenti della classe e dividendola per tutte le parole che occorrono in quella classe.

Un miglioramento di questa stima è

$$P(w|h_i) = \frac{\#\{\text{occorrenze di } w \text{ in } h_i\} + 1}{\#\{\text{occorrenze qualsiasi in } h_i\} + \#\{\text{parole}\}}$$

Apprendere dagli esempi

L'algoritmo «naive Bayes» richiede un «addestramento» su un insieme di documenti la cui classificazione sia nota.

Si tratta cioè di un algoritmo di *machine learning supervisionato*, che per poter esprimere un parere su dati di classificazione ignota deve aver elaborato numerosi esempi classificati.

L'algoritmo ha necessità di un numero sufficiente di documenti classificati per poter stimare le probabilità condizionate con le formule precedenti.

Un esempio classico

Nel nostro caso abbiamo usato il classico repository *20 Newsgroups*, che contiene 20.000 documenti di testo tratti da news degli anni ‘90.

Ci sono 20 categorie di interesse e 1.000 documenti per categoria.

Scegliendo come *training set* 2/3 del corpus e lasciando i restanti come *testing set*, abbiamo ottenuto una percentuale di 80% di classificazioni corrette su esempi che il programma non aveva incontrato prima.

Un esempio classico

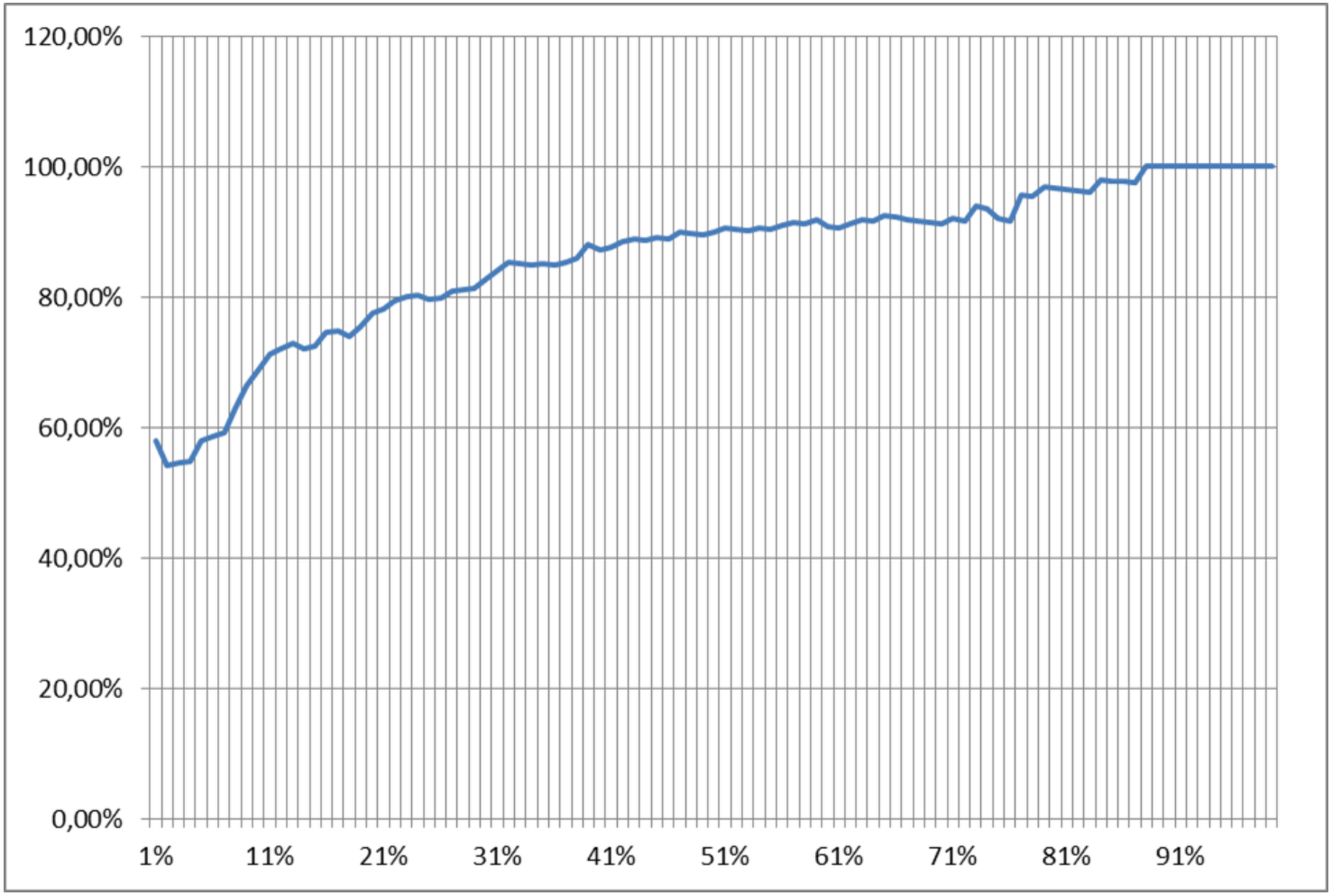
Limitandosi a sole tre classi e 100 file in totale per ogni classe si può mostrare facilmente come varia la percentuale di risposte esatte E rispetto alla percentuale di documenti utilizzati per il training T.

Si verifica che

- se T > 25% allora E > 80%
- se T > 50% allora E > 90%
- se T > 88% allora E = 100%

Nel grafico seguente, sulle ascisse i valori di T, sulle ordinate le risposte esatte corrispondenti.

Un esempio classico



Ecco il programma

Il programma va messo in una cartella che contiene una cartella con i file del dataset, ciascuno in una sottocartella rappresentante la classe corrispondente.

- **nbayes_test.py** svolge il test (chiede in input la percentuale di documenti in una classe usati per il training)
- **nbayes.py** la libreria che calcola naive Bayes
- **20 Newsgroup** dataset su cui provare il programma (che deve stare nella stessa cartella in cui si decomprime il file)

Esempio di utilizzo

Risultato dell'esecuzione del programma nel caso di 4 classi e 200 documenti per classe:

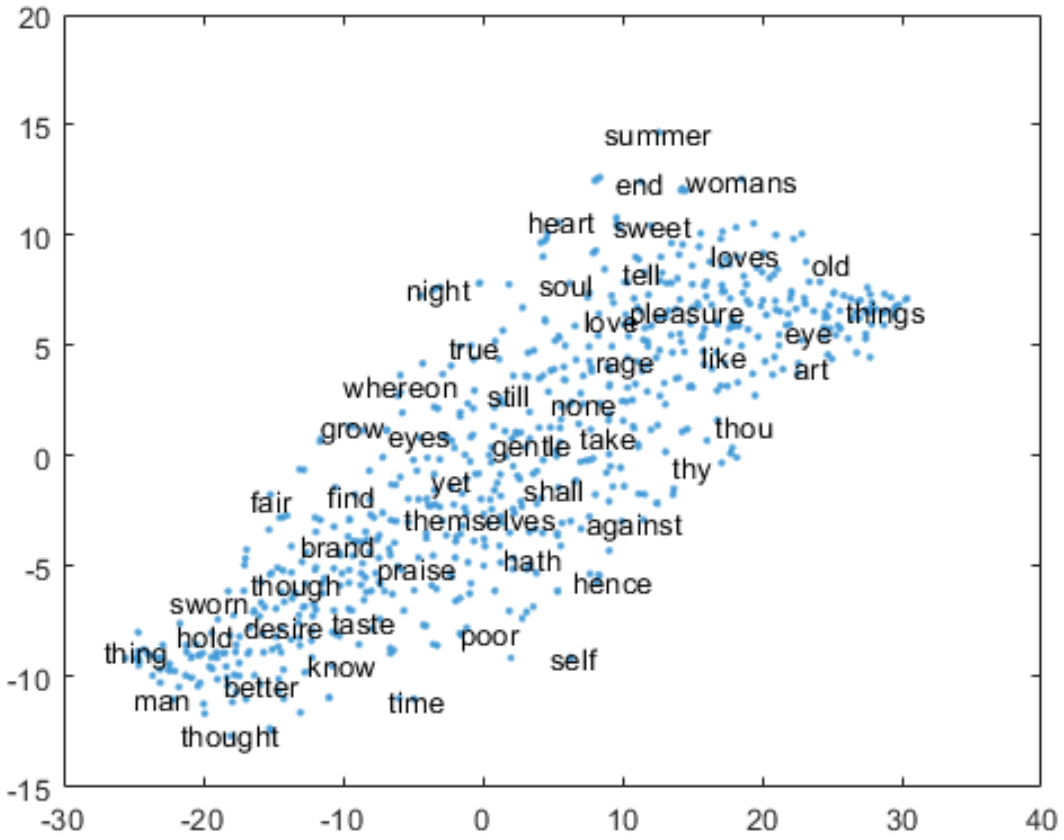
```
Carica nel vocabolario tutte le parole di tutti i documenti
talk.politics.misc
talk.politics.mideast
talk.religion.misc
talk.politics.guns
Corpus con 4 classi e 8044 parole
Percentuale di documenti nel training set: 66

Training set: 66%
Classe talk.politics.misc contiene 19558 parole totali (anche ripetute)
Classe talk.politics.mideast contiene 32907 parole totali (anche ripetute)
Classe talk.religion.misc contiene 16300 parole totali (anche ripetute)
Classe talk.politics.guns contiene 19813 parole totali (anche ripetute)

Fase di test: classifica i documenti che non sono nel training set.
Classe talk.politics.misc 73.13432835820896% di risposte esatte
Classe talk.politics.mideast 81.34328358208955% di risposte esatte
Classe talk.religion.misc 95.5223880597015% di risposte esatte
Classe talk.politics.guns 92.53731343283582% di risposte esatte

Risultato totale della classificazione: 85.63432835820896% di risposte esatte
Premere un tasto per terminare...
```

4. Punti, vettori e parole



Facile come l'algebra...

Joseph Louis Lagrange ebbe a dire del suo amico Antoine Lavoisier che il suo grande merito era stato di rendere la chimica «*facile come l'algebra*».

Gli spazi cartesiani

Cartesio lo aveva fatto con la geometria, rendendola «facile come l'algebra» e sostituendo costruzioni geometriche con calcoli algebrici.

Poiché un punto è determinato dalle sue coordinate (un punto è una sequenza ordinata di numeri), che sono numeri, i punti nello *spazio cartesiano* si possono sommare e sottrarre fra loro agendo sulle singole coordinate…

Questo può farsi in un numero di dimensioni qualsiasi: la dimensione è il numero di coordinate che caratterizzano un punto!

Lo spazio N-dimensionale

Ricordiamoci un po' di geometria cartesiana.

Lo spazio cartesiano di dimensione N è l'insieme $\mathbf{R}^N$ delle *N*-ple ordinate di numeri: nei casi $N = 2$ e $N = 3$ ritroviamo il concetto di piano e spazio cartesiano.

Tutta la geometria euclidea si può «mappare» sullo spazio cartesiano, e le relazioni e le costruzioni geometriche si mappano su relazioni numeriche e calcoli.

Disegno di punti nel piano cartesiano

Punti e vettori

Lo spazio cartesiano $\mathbf{R}^N$ è uno spazio di punti, ciascuno dei quali è individuato univocamente dalle proprie coordinate: la differenza fra due punti *P* e *Q* si chiama un *vettore* ed è intuitivamente lo spostamento che occorre applicare a *P* per ottenere *Q*:

$$v = Q - P = \overrightarrow{PQ}$$

Anche un vettore è rappresentato da una n-pla di numeri, che si possono sommare alle coordinate di un punto per ottenere il punto $P + v$

Cliccare qui per la demo della somma punto-vettore!

Lo spazio vettoriale

L'insieme dei vettori si chiama *spazio vettoriale*, e su di esso sono definite due operazioni "componente per componente" indotte da quelle fra numeri: se $v = (v_1, \ldots, v_N)$, $w = (w_1, \ldots, w_N)$ e $\alpha \epsilon \mathbf{R}$ allora:

$$(v_1, \ldots, v_N) + (w_1, \ldots, w_N) = (v_1 + w_1, \ldots, v_N + w_N)$$
$$\alpha(v_1, \ldots, v_N) = (\alpha\, v_1, \ldots, \alpha\, v_N)$$

Cliccare qui per la demo della somma!

Cliccare qui per la demo del prodotto!

Lunghezza di un vettore

La *lunghezza* (euclidea) di un vettore *v* è il numero non negativo

$$|v| = \sqrt{{v_1}^2 + {v_2}^2 + \cdots + {v_N}^2}$$

Esistono altre possibili distanze (tutte topologicamente equivalenti).

Questa è direttamente ispirata al teorema di Pitagora e corrisponde, nel piano e nello spazio cartesiani, alla usuale distanza euclidea.

Distanza fra due punti

La *distanza* (euclidea) fra due punti dello spazio cartesiano *P* e *Q* è la lunghezza della loro differenza (che come sappiamo è un vettore)

$$d(P,Q) = |P - Q| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_N - y_N)^2}$$

Se due punti hanno distanza minore di *r* allora vuol dire che uno dei due è sempre incluso in ogni "palla" di centro l'altro e raggio maggiore di *r*.

Distanza fra due punti

Correlazione fra due vettori

La *correlazione* (o "similarità") fra due vettori è un numero fra -1 e 1 definito come

$$c(v,w) = \frac{v_1 w_1 + v_2 w_2 + \cdots + v_N w_N}{|v| \times |w|}$$

È del quoziente fra il "prodotto scalare" per il prodotto delle loro lunghezze.

Questo indice si chiama anche (impropriamente) "distanza del coseno".

Distanza e correlazione fra due vettori

Documenti come punti

Le tecniche di «word embedding» provano a rendere l'analisi dei documenti «facile come l'algebra»: una prima idea è

- Scandire tutte le parole di tutti i documenti da analizzare
- Considerare lo spazio cartesiano con *N (*= numero di parole distinte) dimensioni: ciascun asse cartesiano è una parola
- Dato un documento, contare le parole e associare a ogni parola in esso un numero, la sua «coordinata» nello spazio delle parole: un documento è quindi un punto nello spazio!
- Se questa associazione è fatta in modo da tenere conto dell'analisi statistica del testo (es: Bayes) allora punti vicini corrispondono a documenti simili…

Word Embedding:
le parole come punti

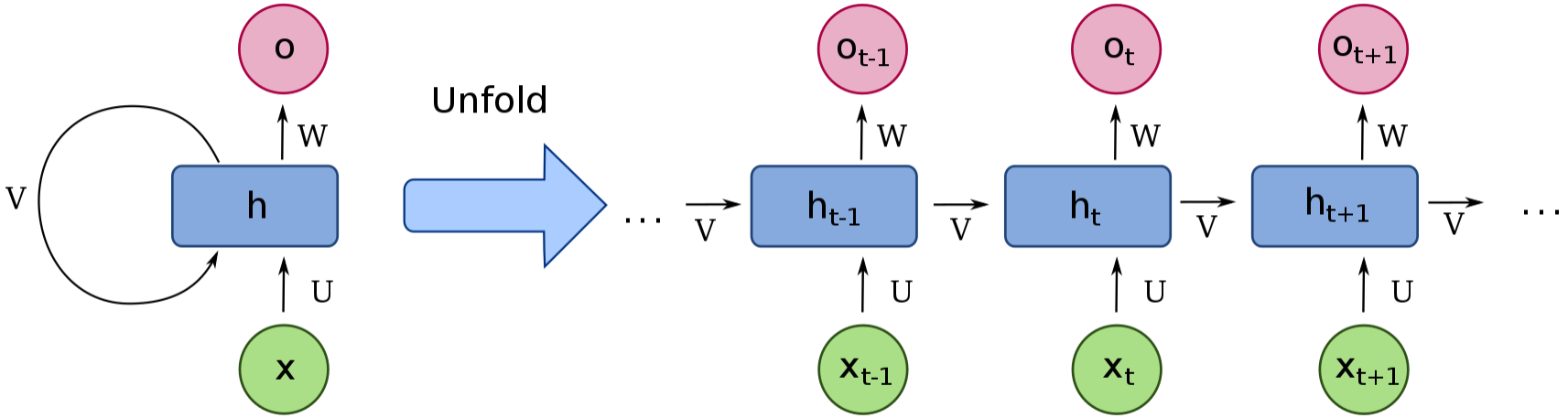
Un uso più recente degli spazi N-dimensionali consiste nell'usare ampie raccolte di documenti per associare a una qualsiasi parola che occorra nel corpus un punto nello spazio di dimensione N fissata.

Occorre trovare una funzione {parole} → {punti} che consenta di associare a parole dal significato simile punti vicini.

Nel fare questo si è scoperto un algoritmo che fa corrispondere alle operazioni algebriche fra punti e vettori operazioni semantiche sul testo: il sogno di Leibniz!!!

Questi algoritmi sono le reti neurali ricorrenti.

5. Digressione: le reti neurali ricorrenti



Idee vecchie, pratiche nuove

Le idee essenziali sul *machine learning* risalgono agli anni '40 (N. Wiener, W. McCulloch), '50 (R. Bellman, F. Rosenblatt) e '60 (M. Minsky), senza voler risalire ai metodi di ottimizzazione di Newton, Leibniz, Bernoulli, Eulero, Lagrange, etc. (XVIII-XIX secc.).

Ma è soltanto dalla fine degli anni ‘90 che la capacità di memoria e calcolo delle macchine consentono di sfruttarne appieno la potenza, e solo dagli anni '10 del XXI secolo che si riescono a implementare architetture "profonde". Questi algoritmi di sono tanto più efficienti quanti più dati riescono a "macinare", e oggi è disponibile una quantità astronomica di dati da elaborare.

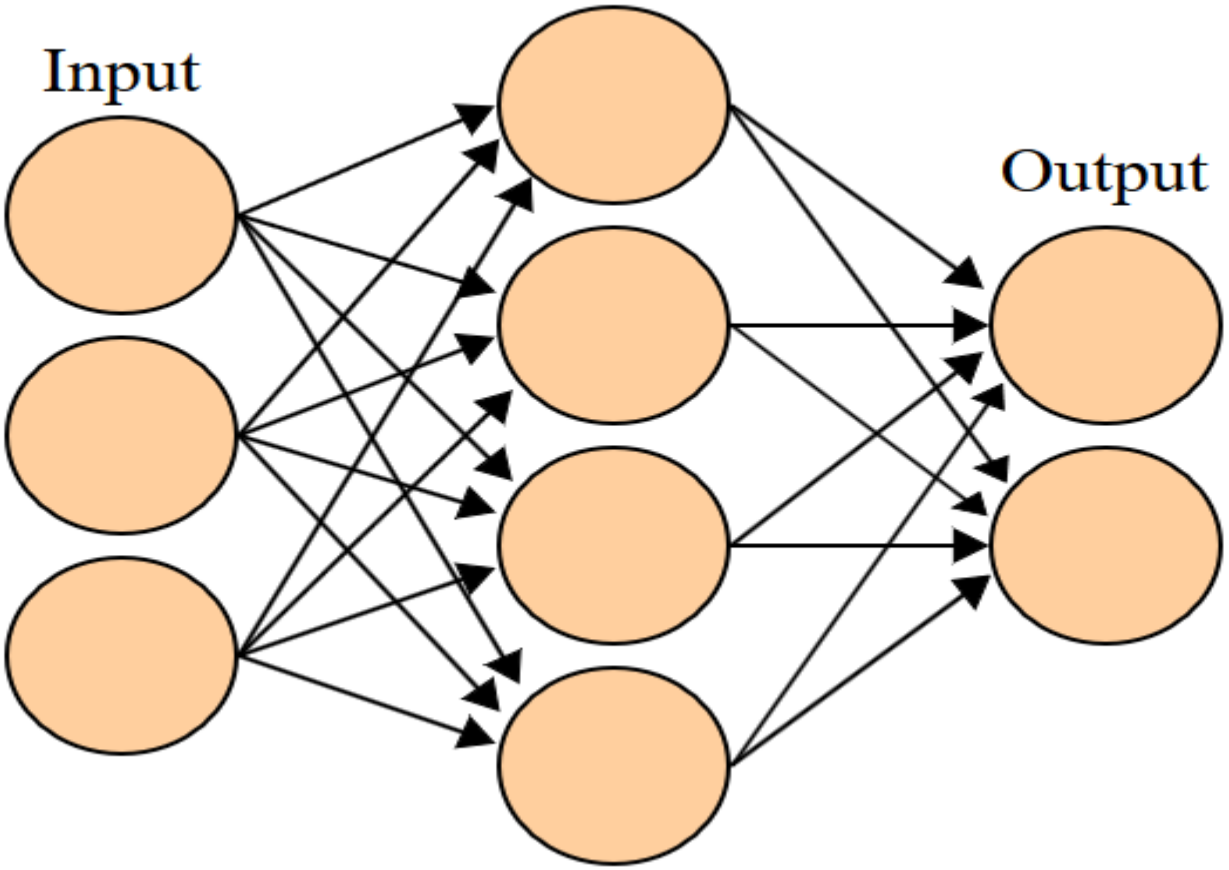
Cosa sono le reti neurali?

Una *rete neurale* è un algoritmo di ottimizzazione che si ispira a un modello iper-semplificato del cervello.

L'idea è di considerare una rete composta da una serie di neuroni disposti in «strati» (*layer*): uno strato di input prende i dati dal «mondo esterno», gli strati interni elaborano l'informazione e uno strato di output emette un risultato.

Ciascuno strato è composto da neuroni che lavorano in parallelo e che sono collegati ai neuroni dello strato precedente e dello strato seguente.

Schema di una rete neurale con uno strato interno



Come calcola un neurone artificiale

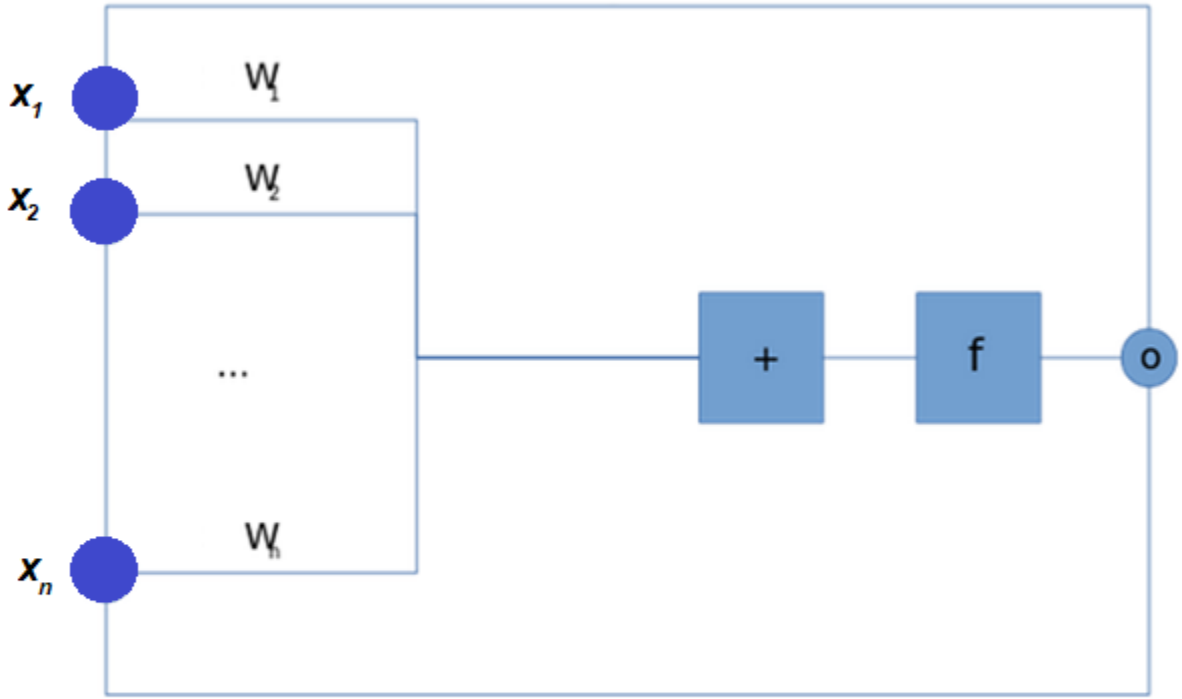
Il modello di un singolo neurone artificiale è stato ideato negli anni 1940 (!) e il suo stato interno è completamente determinato da un vettore $(w_1, w_2, \ldots, w_n)$.

Il neurone accetta in input *n* numeri $x_1, x_2, \ldots, x_n$ e fornisce in output un singolo numero combinando questi numeri come segue:

$$output = f(x_1 w_1 + x_2 w_2 + \cdots + x_n w_n)$$

dove *f* è una funzione non lineare (si noti che pesi e input sono combinati calcolandone la correlazione come vettori).

Modello di un neurone (aka "perceptrone")



$$o = f\left(\sum_{k=1}^{n} x_k \cdot W_k\right)$$

Propagazione fra neuroni

Un neurone si attiva quando le sue terminazioni di input sono stimolate dai neuroni del *layer* precedente oltre una certa soglia.

Una volta attivato, il neurone emette a sua volta un segnale in output che è il risultato di una funzione non lineare applicata ai suoi input.

In questo modo i segnali inviati allo strato di input si propagano all'interno della rete fino allo strato di output: i neuroni di quest'ultimo strato costituiscono il risultato del calcolo effettuato dalla rete.

Algoritmo di calcolo di un rete

I valori dei pesi della rete di neuroni determinano come i singoli neuroni reagiscono agli stimoli di input e quale risultato producono in output: questi valori sono quindi quelli responsabili del calcolo effettivo tramite la rete.

I pesi si possono modificare e si devono modificare, in quanto la rete è generalmente inizializzata a dei valori casuali bassi nei suoi pesi, che sono poi modificati con un algoritmo chiamato «*backpropagation*» in modo da modificarsi per migliorare le proprie prestazioni nel fornire, dato un input, l'output corretto.

Training data

Per far svolgere a una rete un calcolo ben determinato, le si sottopongono dei *training set*, cioè insiemi di input per i quali si sa la risposta attesa, e vedere cosa risponde la rete: se la risposta della rete non è quella attesa si modificano i pesi della rete in modo da "costringerla" la prossima volta a rispondere in modo corretto per quel determinato input.

Disponendo di un training set abbastanza esteso, la rete potrà addestrarsi a riconoscere correttamente una gran varietà di input. Una volta addestrata, potrà essere usata su input per i quali la risposta non è nota.

Applicazioni delle reti

Qualsiasi problema esprimibile come approssimazione di una funzione ignota a partire da alcuni valori noti che assume può essere risolto con una rete neurale.

Poiché qualsiasi informazione rappresentabile in un computer viene digitalizzata (trasformata in numeri) è chiaro come le reti neurali possano essere impiegate per riconoscere o anche generare immagini, suoni e testi.

In particolare, come abbiamo visto, un documento di testo può essere trasformato in un vettore, quindi può essere elaborato da una rete neurale.

Training supervisionato e non

Se trasformiamo l'output della rete in un valore di probabilità (e ci sono varie tecniche per farlo partendo da una rete che emette semplicemente dei numeri in output), allora possiamo anche addestrarla in modo «non supervisionato» *(unsupervised).*

Più precisamente potremmo produrre un training set facendo *sampling* da una distribuzione piuttosto che etichettando manualmente i singoli documenti del training set: questo è l'approccio dell'algoritmo di *word embedding* che vogliamo descrivere.

Reti ricorrenti

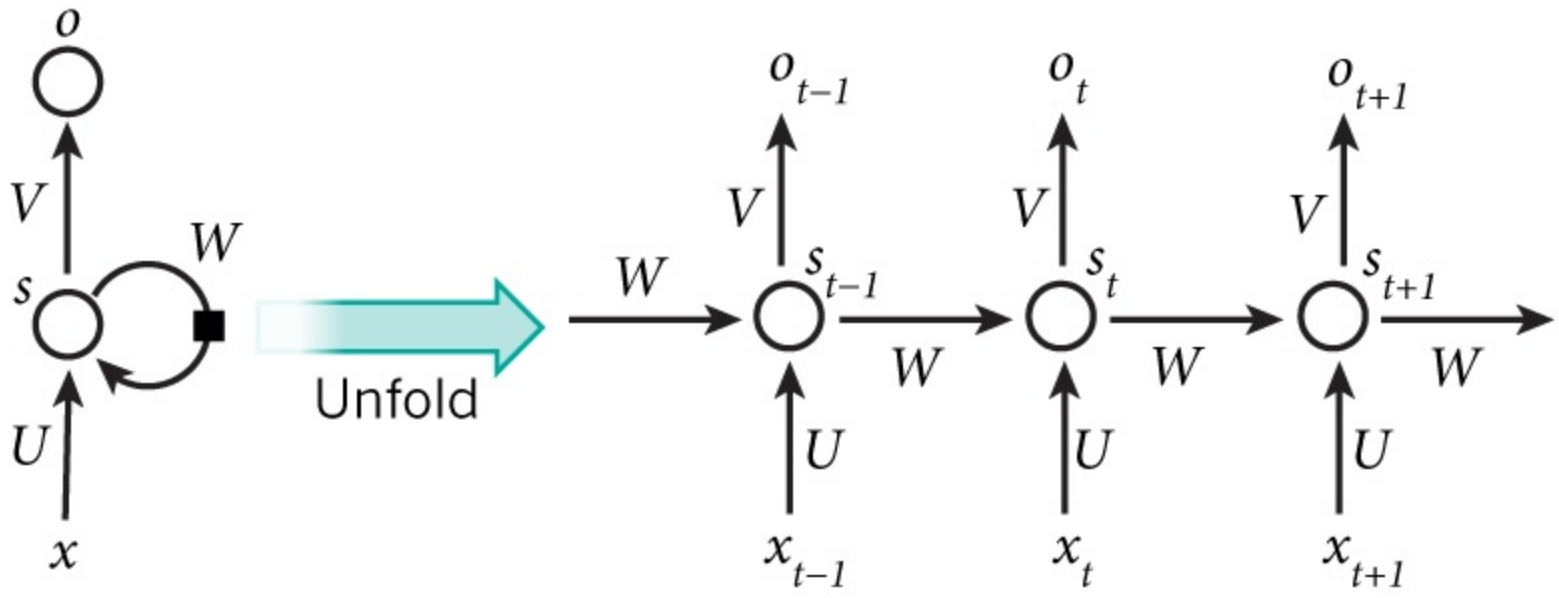
Le reti neurali «classiche» non hanno memoria, nel senso che il valore dei loro pesi al tempo t_1 dipende soltanto dal valore che avevano al tempo t_0 e, ovviamente, dall'ultimo *training example*.

Una *rete ricorrente* ha invece alcuni neuroni al suo interno che forniscono un ulteriore input alla rete e che sono alimentati dall'output: questi neuroni formano una sorta di «memoria storica» di quanto accaduto alla rete.

Infatti, queste reti sono usate con successo nella predizione delle serie storiche.

Reti ricorrenti

Una rete ricorrente con tre neuroni: uno di input, uno di output e uno interno che si autoalimenta. A destra un suo *unfolding* in due istanti.

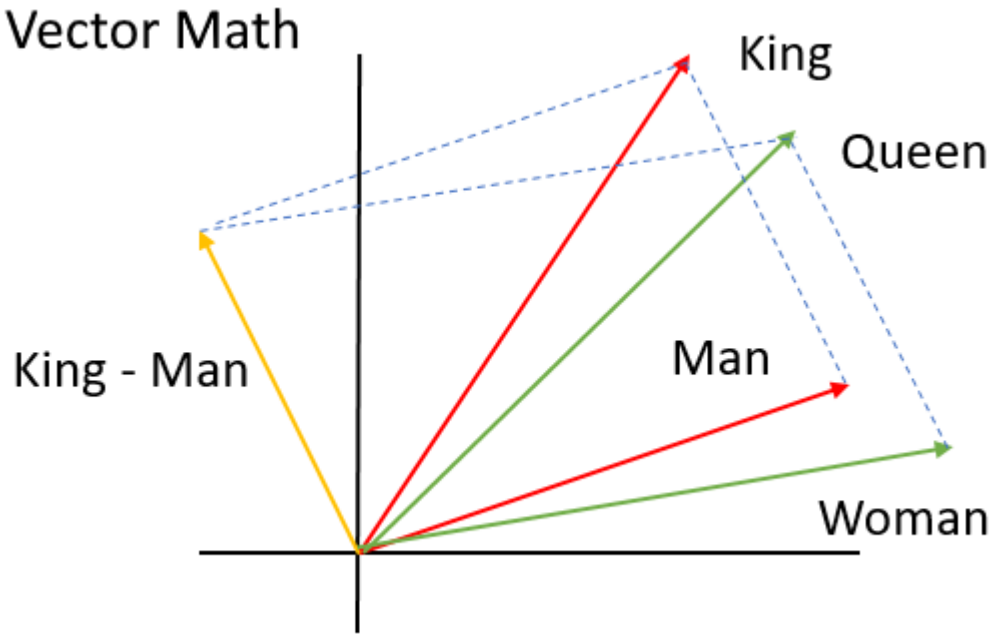


Reti ricorrenti e NLP

Le reti ricorrenti sono state impiegate con molto successo nella comprensione e anche nella generazione di testi: pensando ai nostro esempio elementare (markoviano) possiamo immaginare quanto un modello non lineare come una rete ricorrente possa migliorare le prestazioni di sistemi di generazione di testi e di classificazione e comprensione dei testi.

Combinando comprensione, generazione e il fatto che questi sistemi non dipendono dalla lingua, è chiaro il perché siano utilizzati per la traduzione automatica da una lingua a un'altra.

6. L’algebra lineare di parole e concetti



Generalizzazione dei modelli markoviani

L'idea del modellino markoviano visto in precedenza è di modellizzare statisticamente le relazioni fra coppie di parole consecutive in un testo.

Più in generale potremmo, dato un testo le cui parole, nell'ordine e anche con ripetizioni, sono

$$p_1 \ p_2 \ p_3 \ p_4 \ p_5 \ \dots \ p_N$$

scorrere tutte le parole (per i = 1, 2, ..., N) e considerare per ciascuna parola le k che la precedono e le k che la seguono:

Esempio

Esempio (con k = 2):

"Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene **non** interrotte di monti, tutto a seni e a golfi, a seconda dello sporgere e del rientrare di quelli, vien, quasi a un tratto, a ristringersi, e a prender corso e figura di fiume, tra un promontorio a destra, e un'ampia costiera dall'altra parte"

Esempio

Esempio (con k = 2):

"Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene non **interrotte** di monti, tutto a seni e a golfi, a seconda dello sporgere e del rientrare di quelli, vien, quasi a un tratto, a ristringersi, e a prender corso e figura di fiume, tra un promontorio a destra, e un'ampia costiera dall'altra parte"

Esempio

Esempio (con k = 2):

"Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene non interrotte **di** monti, tutto a seni e a golfi, a seconda dello sporgere e del rientrare di quelli, vien, quasi a un tratto, a ristringersi, e a prender corso e figura di fiume, tra un promontorio a destra, e un’ampia costiera dall’altra parte"

Esempio

Esempio (con k = 2):

"Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene non interrotte di **monti**, tutto a seni e a golfi, a seconda dello sporgere e del rientrare di quelli, vien, quasi a un tratto, a ristringersi, e a prender corso e figura di fiume, tra un promontorio a destra, e un'ampia costiera dall'altra parte"

Esempio

Esempio (con k = 2):

"Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene non interrotte di monti, **tutto** a seni e a golfi, a seconda dello sporgere e del rientrare di quelli, vien, quasi a un tratto, a ristringersi, e a prender corso e figura di fiume, tra un promontorio a destra, e un'ampia costiera dall'altra parte"

Esempio

Esempio (con k = 2):

"Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene non interrotte di monti, tutto **a** seni e a golfi, a seconda dello sporgere e del rientrare di quelli, vien, quasi a un tratto, a ristringersi, e a prender corso e figura di fiume, tra un promontorio a destra, e un'ampia costiera dall'altra parte"

Esempio

Esempio (con k = 2):

"Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene non interrotte di monti, tutto a **seni** e a golfi, a seconda dello sporgere e del rientrare di quelli, vien, quasi a un tratto, a ristringersi, e a prender corso e figura di fiume, tra un promontorio a destra, e un'ampia costiera dall'altra parte"

Esempio

Esempio (con k = 2):

"Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene non interrotte di monti, tutto a seni **e** a golfi, a seconda dello sporgere e del rientrare di quelli, vien, quasi a un tratto, a ristringersi, e a prender corso e figura di fiume, tra un promontorio a destra, e un'ampia costiera dall'altra parte"

Esempio

Esempio (con k = 2):

"Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene non interrotte di monti, tutto a seni e **a** golfi, a seconda dello sporgere e del rientrare di quelli, vien, quasi a un tratto, a ristringersi, e a prender corso e figura di fiume, tra un promontorio a destra, e un'ampia costiera dall'altra parte"

Il contesto di una parola...

L'algoritmo appena «abbozzato» può essere usato per determinare un modello semantico delle parole di un corpus, considerando delle «finestre mobili» di parole costruite attorno alle parole da analizzare.

L'idea è che il contesto di una parola, che ne può determinare il significato, sono le parole che occorrono nelle «vicinanze» di essa in un testo (per esempio fra le 5 parole che la precedono e le 5 che la seguono).

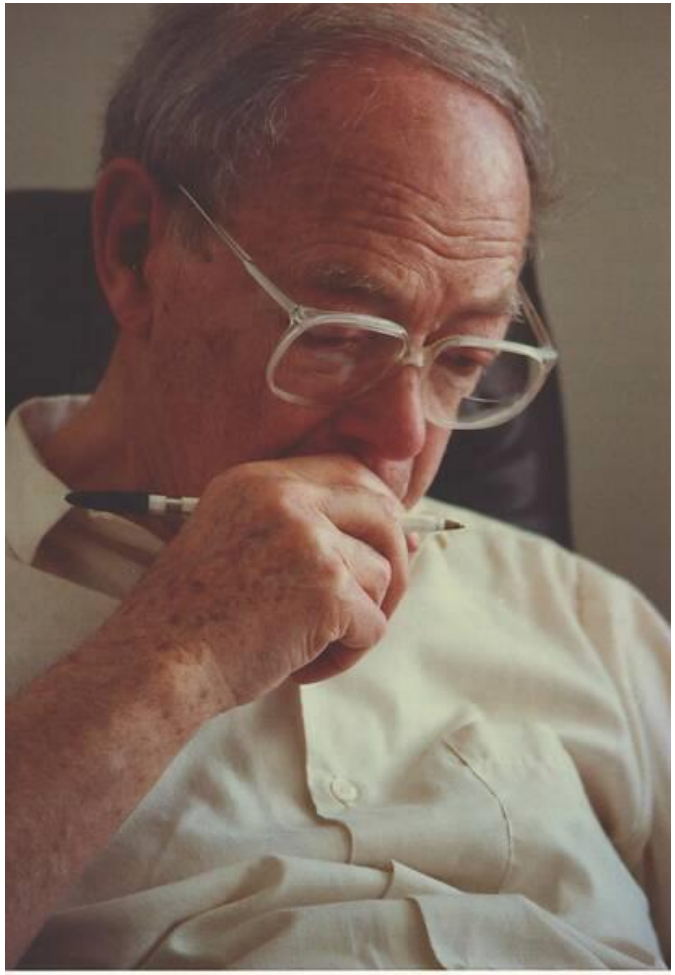
Come sempre accade, questa idea non è recente e non l'hanno avuta gli informatici...

Fondamenti teorici dell'analisi semantica distribuita

«Una parola è caratterizzata dalla compagnia in cui si trova.»

John Rupert Firth

Fondamenti teorici dell'analisi semantica distribuita



«La correlazione fra linguaggio e significato è molto più grande quando consideriamo un discorso connesso. Nella misura in cui la struttura distribuzionale si può scoprire in un discorso, essa è in qualche modo correlata con la sostanza di quanto viene detto; questo è evidente in special modo nei discorsi scientifici.»

Zellig Harris

Word2Vector

Vediamo un esempio ormai classico di «word embedding» che usa la semantica distribuzionale: l'algoritmo *Word2Vector*.

Si tratta di una rete neurale ricorrente con un layer interno, che viene usata in una modalità non supervisionata.

Per ogni parola *p* del corpus si produce un punto *v*(*p*) nello spazio a N dimensioni, dove *N* è la cardinalità *N* del corpus, e che rappresenta la parola *p*. Questi punti (= sequenze di *N* numeri) sono prodotti dall'algoritmo nel corso dell'analisi distribuzionale dell'intero corpus.

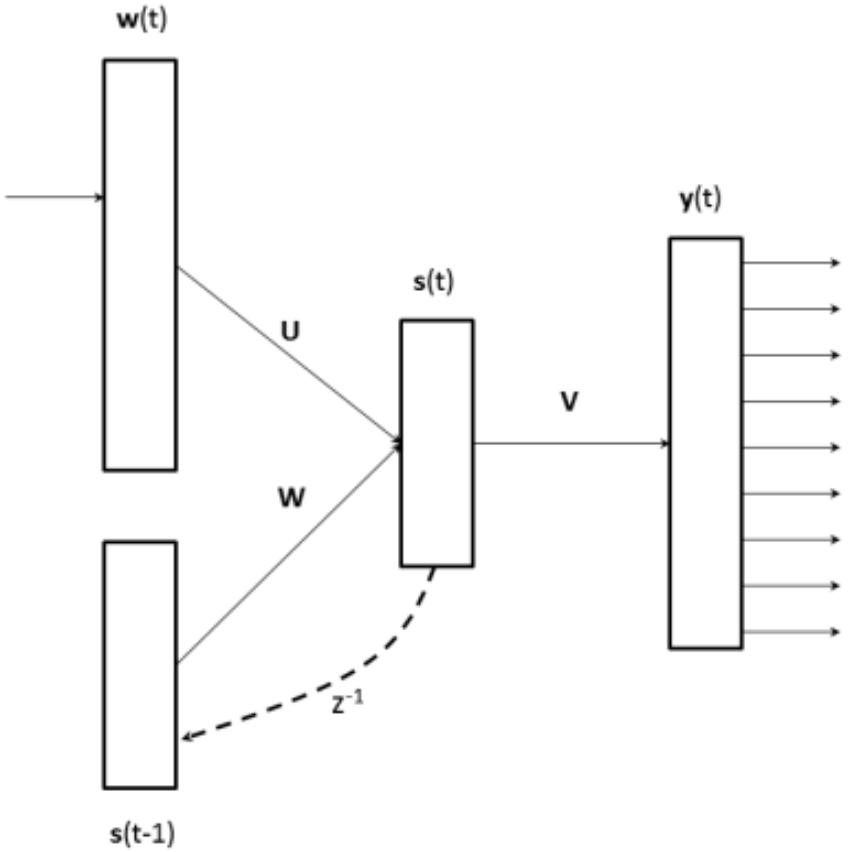
Caratteristiche di Word2Vector

La rete impara il legame che esiste fra la distribuzione di probabilità di una parola e il contesto distribuzionale nel quale appare.

Si noti che l'approccio markoviano di dedurre una parola data la precedente è un caso lineare e particolare di questi due.

Il *layer* interno della rete neurale accumula dei valori prodotti via via che le parole del testo sono presentate alla rete, e conserva quindi memoria dei contesti al cui interno si manifestano le parole esaminate dalla rete: questo è l'elemento cruciale dell'algoritmo.

Architettura della rete

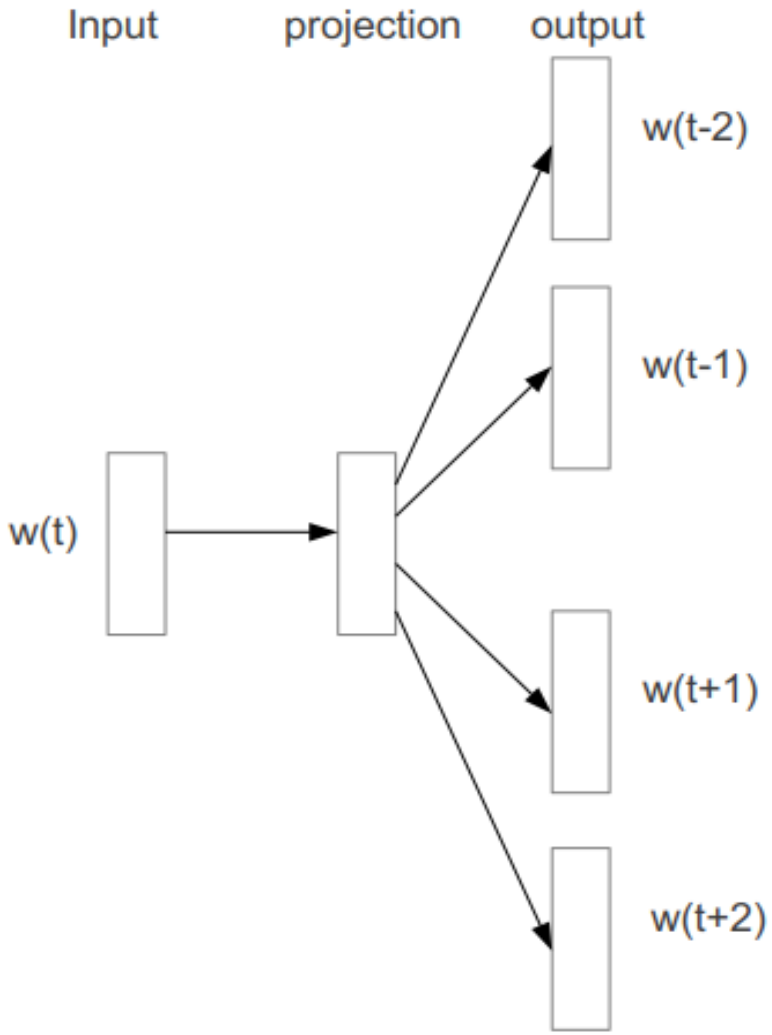


$s(t) = Uw(t) + Ws(t-1)$

$y(t) = Vs(t)$

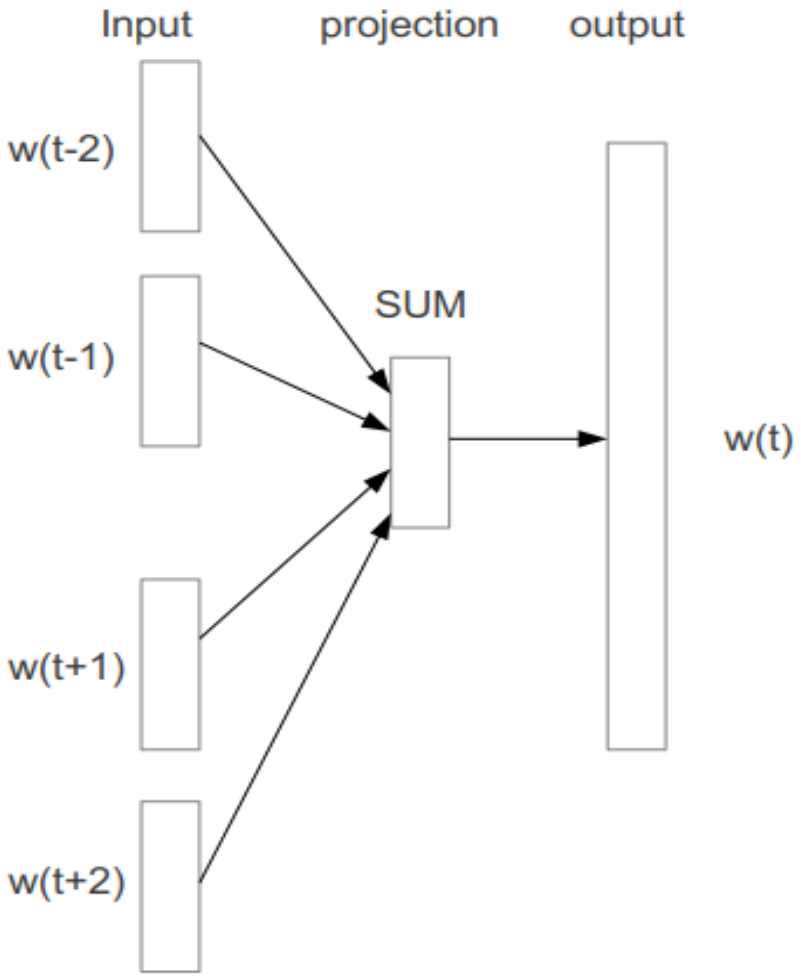
Sia y che s sono filtrati tramite delle funzioni: con la rappresentazione *soft-max* il primo, con una sigmoide $f(z)$ il secondo.

Addestramento Skip Gram



La rete si addestra a produrre la distribuzione di probabilità *y* in modo che dalla parola di input si possa predire il contesto

Addestramento CBOW



La rete si addestra a produrre la distribuzione di probabilità *y* in modo che dal contesto di input si possa predire la parola (cioè si completi il contesto con una parola plausibile).

I punti associati alle parole

I punti associati alle parole, cioè il reale output che interessa per il modello semantico distribuito, sono estratti come colonne dalla matrice *U* che fornisce il collegamento fra il *layer* di input e il *layer* interno della rete neurale.

Questo algoritmo può essere implementato in modo efficiente e, per corpora abbastanza vasti (almeno miliardi di parole), produce risultati molto accurati migliorando di molto le prestazioni dei precedenti algoritmi di analisi semantica distribuita.

Esempio d'uso

Proviamo a visualizzare i punti che *word2vec* associa alle parole: per farlo utilizziamo uno spazio con $N = 200$ dimensioni che conterrà le circa 70K parole del corpus, elaborate dalla rete neurale su un corpus di documenti presi da pagine di *Wikipedia* (en).

Nella seguente demo inserire una lista di parole che il programma assocerà i punti corrispondenti, proiettandoli sul piano cartesiano (con l'algoritmo t-SNE che tenta di preservare le distanze):

Cliccare qui per la demo

Punti = parole
Vettori = concetti

Un aspetto che ha del miracoloso di questo algoritmo è che alcune relazioni semantiche si ottengono con operazioni fra i vettori delle parole corrispondenti.

Per esempio: se scriviamo *v(p)* per il punto associato alla parola *p*, e *p(v)* per la parola corrispondente al *punto più vicino* al punto *v*, troviamo che

p(v("king") - v("man") + v("woman")) = "queen"

Cioè il vettore "king" - "man" corrisponde al concetto di «regalità"...

Algebra della semantica

Altri esempi portati dai creatori di word2vec su un dataset 1000 volte più grande del nostro:

p(v("bigger") - v("big") + v("cold")) = "colder"
p(v("Paris") - v("France") + v("Italy")) = "Rome"
p(v("sushi") - v("Japan") + v("Germany")) = "bratwurst"
p(v("Windows") - v("Microsoft") + v("Google")) = "Android"

Provare per credere!!!

Parole simili

Un altro semplice esperimento che è possibile provare è, data una parola, trovare quelle che le sono più vicine, per esempio le 10 più vicine:

Trova le parole simili a una parola data

Con questo metodo possiamo anche trovare dei bias in un corpus, per esempio vedendo cosa è vicino a “man” e cosa è vicino a “woman”, etc.

7. Capire il contesto

Oltre il word embedding

Per quanto i risultati dell'algoritmo *Word2Vector* (e delle sue generalizzazioni) siano sorprendenti e molto soddisfacenti per certe applicazioni, l'associazione di un punto a una parola (e quindi di un vettore a una coppia di parole) è fatta in base al contesto della parola nelle occorrenze del corpus ma una volta fatta rimane sempre la stessa.

In altre parole, il punto associato alla parola non cambia a meno di modificare il corpus dal quale è calcolato, per esempio aggiungendo nuovi testi. Si tratta dunque di una «codifica statica».

L’idea dei «transformer»

Una idea molto recente (risale al 2017) per trattare in modo molto efficace il contesto delle parole è quella dei *transformer*.

Questi algoritmi prevedono che il testo sia prima codificato in modo da enucleare le relazioni più significative fra le parole, e poi decodificato per calcolare infine le probabilità di una parola dato un contesto come negli algoritmi precedenti.

L’architettura è piuttosto complicata e non è il caso di riportarla qui: vediamone solo alcune caratteristiche e un esempio.

BERT

BERT è acronimo di *Bidirectional Encoder Representations from Transformers.*

Di nuovo questo algoritmo è stato inventato nei laboratori di Google, nel 2018.

Bidirezionale in quanto il contesto di una parola è preso sia dalle parole che la precedono che da quelle che la seguono;

Encoder perché utilizza il tipico schema dei transformer, che infatti sono anche citati nell'acronimo.

BERT: bidirezionalità

Un sistema bidirezionale va a “mascherare” alcune delle parole nelle frasi del corpus per addestrarsi a predirle: per esempio nella frase

Arrivederci e grazie per tutto il pesce

potremmo mascherare *tutto* e *pesce*, ottenendo lo schema

Arrivederci e grazie per X il Y

Abbiamo così una frase cui associamo una label (X=*tutto*, Y=*pesce*) creando in questo modo un *training record*.

BERT: correlazione di frasi

Naturalmente in questo modo possiamo creare moltissime combinazioni e quindi un training set smisurato sul quale apprendere a predire le parole dato il contesto.

Un altro meccanismo di BERT consiste nel predire se due frasi sono collegate. Per esempio *Non temo l'intelligenza della macchina. Temo la stupidità dell'Uomo* sono due frasi chiaramente correlate, mentre *Non temo l'intelligenza della macchina. Che tempo fa?* non lo sono.

Per ottenere questo risultato BERT si esercita su enormi moli di frasi correlate e non.

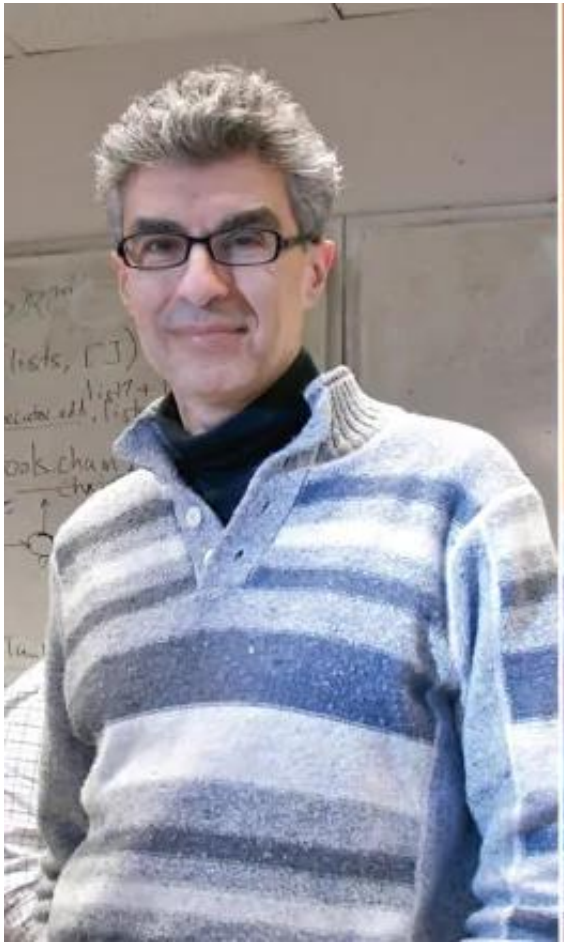
BERT: deep learning

Le reti neurali che abbiamo considerato fin qui non sono “profonde”: BERT è invece costretto a utilizzare il cosiddetto *deep learning* che prevede reti che abbiano centinaia di strati interni e un numero astronomico di neuroni.

Oltre a questo queste reti di deep learning utilizzano algoritmi particolari e accorgimenti nel passare da un layer all’altro per evitare i problemi di performance che affliggevano le reti negli anni ‘90.

Queste idee hanno fruttato ai loro scopritori il premio Turing nel 2018

Deep Learning Godfathers



Yoshua Bengio Geoffrey Hinton Yann LeCun

BERT: non si addestra in casa

Le caratteristiche di BERT (e degli algoritmi transformer in generale) richiedono che per addestrarli sui voluminosi training set sia richiesta una strumentazione particolare, cioè macchine potenti che consentano calcoli paralleli e siano in grado di svolgerli velocemente.

Per questo BERT separa nettamente la fase di *training* da quella di *tuning* che consiste nel trovare la configurazione giusta dell'algoritmo per uno specifico problema: per la prima ci si affida a un pre-training svolto da Google sui suoi server, la seconda ce la possiamo fare in casa.

Vediamo un semplice esempio.

Esempio: *sentiment analysis*

Tentiamo un semplice esperimento di *sentiment analysis*, cioè capire se una frase sta dicendo qualcosa di positivo o di negativo.

Per farlo consideriamo un celebre dataset di recensioni di film prese dal sito *Rotten Tomatoes*. Si tratta di un database di 10605 giudizi, molto brevi (la maggior parte di una sola frase) su dei film, e un sentiment ranking per ognuno di essi, cioè un punteggio fra 0 e 1 tanto più alto quanto il giudizio è positivo e tanto più basso quanto è negativo

Esempi

Alcune frasi del training set e relativi punteggi:

“This is a movie where the most notable observation is how long you 've been sitting still” [0.27778 😫]

“this one is a sweet and modest and ultimately winning story” [0.84722 ☺]

“and not worth” [0.43056 😐]

Usare un modello “*pretrained*”

Per semplicità lavoreremo solo sulle singole frasi e utilizzeremo un modello BERT pre-addestrato, adattandolo alle esigenze del problema: ne scegliamo uno semplice, "BERT-tiny". Questo modello molto piccolo vuole in input 64 numeri e ne restituisce 128 (e internamente ha 4.369.152 parametri!).

La codifica del risultato del modello BERT per determinare il sentiment la faremo con una semplicissima rete neurale un layer di dropout e un layer denso da 128 neuroni di input per un totale di 16512 parametri interni

Usare il modello

Ho salvato la rete neurale su un file per caricarla “al volo” e farla provare cliccando sul link seguente:

Toy Sentiment Analysis

Questo programma usa la libreria Bert: una implementazione più complessa ma molto più precisa sullo stesso dataset usando la libreria TensorFlow in modo esplicito si può trovare qui:

Classify text with Bert - Google

Grazie per l'attenzione!!!

Credits
Wikipedia per le immagini.
Tomas Mikolov per l'algoritmo, il codice e i paper di riferimento (da alcune sue slide abbiamo preso delle figure).
Google Code per il codice di Word2Vector.
Jason Rennie per il dataset utilizzato per naive Bayes.
Matt **Mahoney** per il dataset utilizzato per word2vec.
R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng and C. Potts per il dataset sulla sentiment analysis
Google Research per il modello tiny-BERT

http://www.caressa.it
https://www.linkedin.com/in/paolocaressa
@www_caressa_it