MarTech

Text analysis in Python

NLP

Ing. Paolo Fantozzi

Modello di linguaggio

Per arrivare a “comprendere” in modo automatico un testo si deve prima creare un modello che lo rappresenti

Modello di linguaggio

Un **modello di linguaggio** può essere considerato come una distribuzione di probabilità.

Qual è la probabilità di avere una data parola, sapendo le *n-1* parole precedenti?

Distribuzione di probabilità

Come calcolare tale probabilità?

Analizziamo i dati di cui siamo a disposizione e le occorrenze di quella parola.

N-grammi

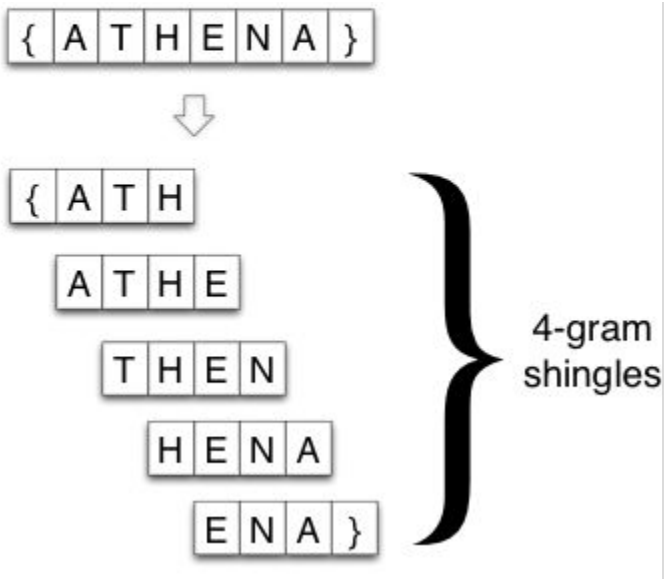
Data una frase contenente *N* parole, un **n-gramma** di tale frase è una sua qualsiasi sotto-sequenza ordinata di *n*<*N* parole

This is Big Data AI Book

Uni-Gram	This	Is	Big	Data	AI	Book
Bi-Gram	This is	Is Big	Big Data	Data AI	AI Book	
Tri-Gram	This is Big	Is Big Data	Big Data AI	Data AI Book		

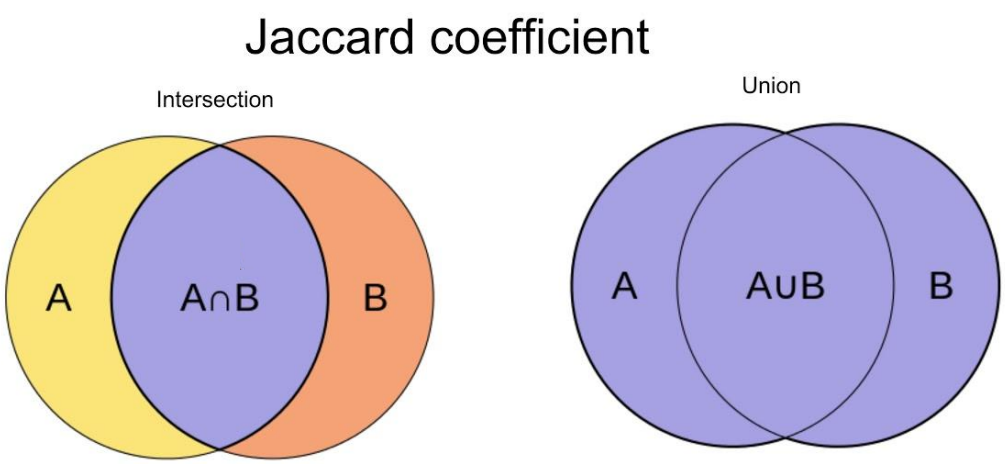
Shingles

Se facciamo scorrere una finestra di lunghezza fissata sui **caratteri** di un documento, il risultato è una serie di ngrams. Questo insieme di ngrams può essere usato per rappresentare il testo



Jaccard similarity

La Jaccard similarity è una misura di similarità tra insiemi



$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Jaccard similarity sulle shingles

Possiamo trovare documenti “simili” attraverso la Jaccard similarity tra le shingles

Dataset

Scaricare ed utilizzare il dataset composto da due file presente al seguente link

http://bit.ly/text-python-amz-revw

Ognuno dei due file è nella forma:

```
__label__1 title_a: review_a
__label__2 title_b: review_b
__label__1 title_c: review_c
```

Le label corrispondono a:

- 1 -> recensione 1 o 2 stelle
- 2 -> recensione 4 o 5 stelle

Jaccard similarity sulle shingles

Trovare i due documenti più simili nel dataset per una size uguale a 3 (file test.txt)

Confrontare solo il testo delle prime 5000 review.

Jaccard similarity sulle shingles

file: 20_shingles_jaccard.py

Bag of n-grams

Così come rappresentiamo un testo come un insieme non ordinato di parole, esso può essere rappresentato come un insieme non ordinato di n-grammi

N-grammi come modello di linguaggio

Gli n-grammi possono essere utilizzati come modello di linguaggio, creando un modello di Markov, normalizzando i gruppi di n-grammi.

$$P(a,b,c) = \frac{C(a,b,c)}{\sum_i C(a,b,i)}$$

N.B. Le prime parole delle frasi si considerano precedute da un token speciale “<s>” n-1 volte, mentre le ultime si considerano seguite da un token speciale “</s>” n-1 volte.

N-grammi come modello di linguaggio

Per calcolare la probabilità di una sequenza arbitraria di parole, si utilizza una catena di probabilità moltiplicando n-gram dopo n-gram in una finestra scorrevole sul documento.

Il problema in tale processo è la precisione nella rappresentazione dei numeri nel software. Pertanto, si utilizza l'esponenziale della somma dei logaritmi delle probabilità.

Laplace Smoothing

Potrebbe esserci la possibilità che un n-gram non compaia mai, portando il prodotto a 0. Per evitare ciò, si aggiunge 1 ad ogni occorrenza di ogni n-gram per ogni classe (Laplace smoothing)

$$P(a,b,c) = \frac{C(a,b,c)+1}{\sum_i (C(a,b,i)+1)}$$

Quale *n* scegliere?

Qual è la dimensione ottima degli n-grammi da scegliere?

n troppo piccolo: perdiamo la causalità tra le parole

n troppo grande: sparsità (oltre che troppa memoria occupata)

Test

Si costruisce il modello su un training set per ogni valore di *n* in un intervallo, e si testa tale modello su di un test set. La probabilità maggiore così ottenuta corrisponderà alla dimensione ottimale.

A parità di probabilità tra due valori di *n*, è sempre vantaggioso considerare il valore più piccolo.

Quale *n* scegliere?

Creare dei modelli basati su n-grams **(su parole)** con *n* uguale a 2 e 3 sul file train. Calcolare successivamente le probabilità sul file test. **Utilizzare solo le prime 10 000 reviews del train set.**

Considerare solo la media delle probabilità calcolate sui casi di test.

Utilizzare tutte le review come lowercase.

Utilizzate spacy per tokenizzare nel seguente modo

```
from spacy.lang.en import English
nlp = English()
tokenizer = nlp.tokenizer
```

Inoltre utilizzare deque per tenere la finestra mobile di parole

```
from collections import deque
```

Quale *n* scegliere?

file: 21_ngrams_model_size.py

Perplexity

Solitamente, per verificare la “bontà” di un modello di linguaggio si utilizza la **Perplexity**, calcolata, a partire dalla probabilità di ogni documento p_d, come:

$$\begin{aligned}\text{Perplexity}(\text{C}) &= \sqrt[N]{\frac{1}{\prod_{i=1}^{m} p(s_i)}} \\ &= 2^{\log_2 \left[\prod_{i=1}^{m} p(s_i)\right]^{-N}} \\ &= 2^{-\frac{1}{N}\log_2 \left[\prod_{i=1}^{m} p(s_i)\right]} \\ &= 2^{-\frac{1}{N}\sum_{i=1}^{m} \log_2 p(s_i)}\end{aligned}$$

Il modello migliore è quello con perplexity più bassa.
Dove: N è la dimensione del dizionario, m la dimensione del corpus e s_i l’i-esimo documento del corpus

Perplexity

Scegliere il valore di *n* ottimale in base alla perplexity

Perplexity

file: 22_ngrams_perplexity.py

Sentiment analysis

La **Sentiment analysis** è un task di classificazione: si cerca di capire quale sia la categoria di un testo

In questo caso le categorie sono: positivo, negativo

Sentiment analysis

Per calcolare le probabilità nel training set si esegue:

- Il calcolo della probabilità per ognuna delle classi
- Il calcolo della probabilità di un certo n-gram in una classe

La probabilità di un n-gram per una classe è il numero di occorrenze di quel n-gram per quella classe, diviso il numero totale di n-gram nella medesima classe

Sentiment analysis

Per calcolare la probabilità che un documento sia in una classe, vengono moltiplicate tra loro le probabilità dei suoi n-gram in quella classe (usando i trucchi matematici già visti in precedenza)

Infine, vengono ignorati tutti gli n-gram che non compaiono nel dataset.

Sentiment analysis

Utilizzare il file train.txt per creare il modello e poi applicarlo sul file test.

Misurare la percentuale di documenti correttamente categorizzati (accuracy)

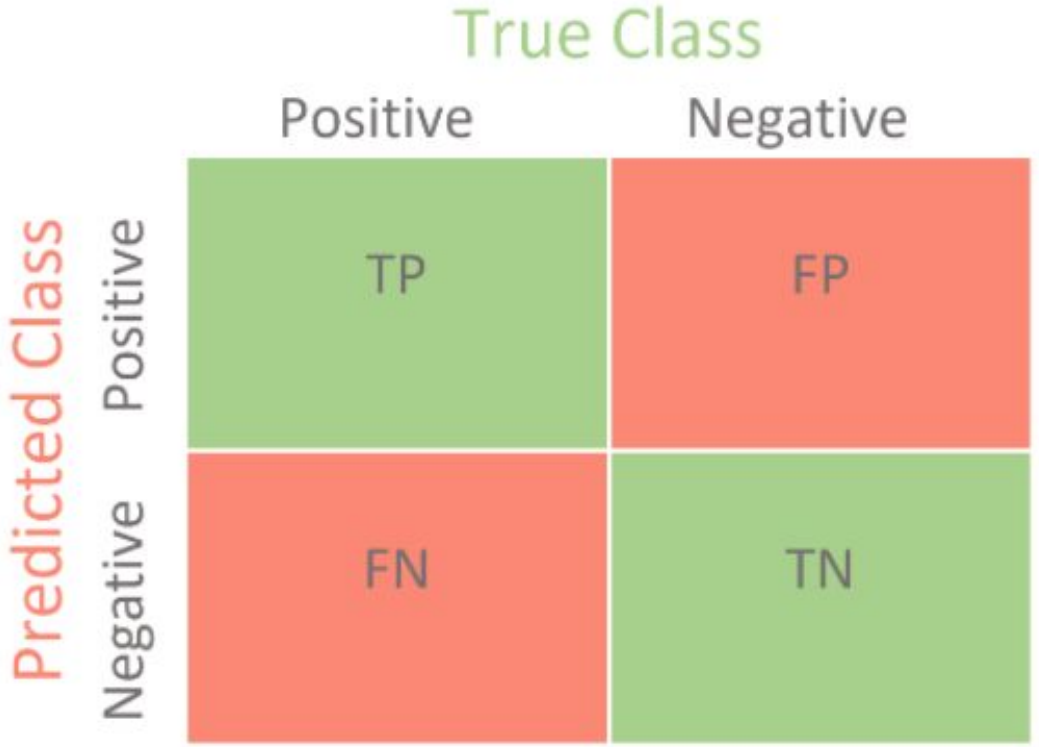
Sentiment analysis

file: 23_ngrams_sentiment.py

Confusion matrix

Nella classificazione si può parlare di veri positivi, veri negativi, falsi positivi e falsi negativi andando a comparare i risultati ottenuti con il classificatore durante, i test a fronte dei valori che si dovrebbero ottenere realmente.

Sulla base di tali percentuali, può essere costruita la **Confusion Matrix**.



Precision

La **Precision** è definita come il rapporto tra i true positive e la somma dei true positive e false positive

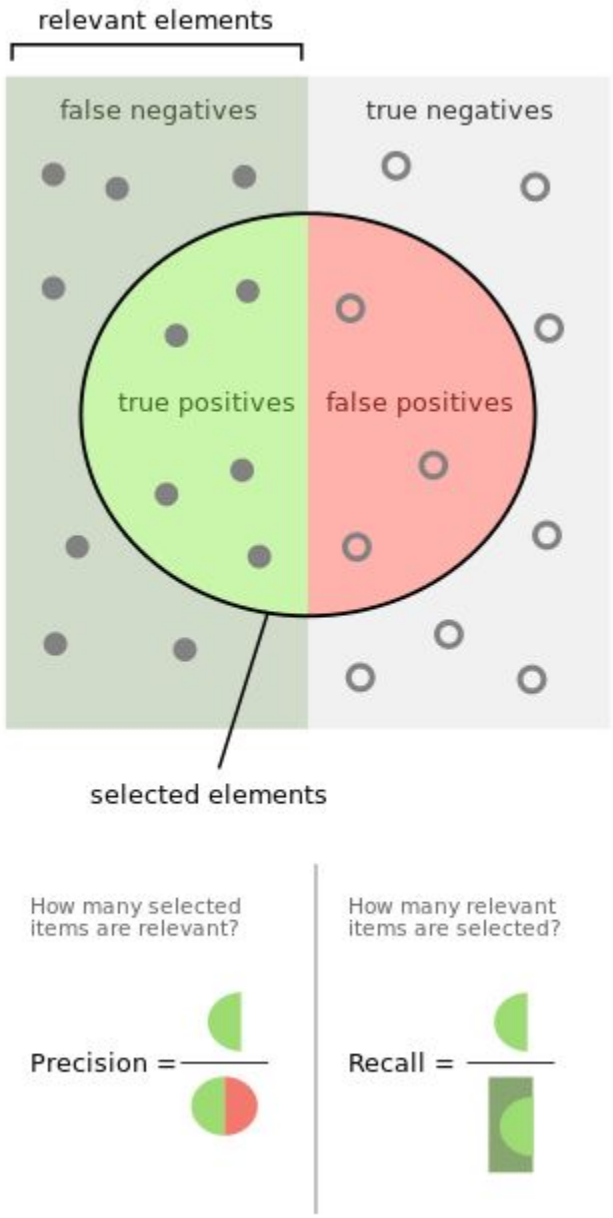
$$P = \frac{tp}{tp + fp}$$

Recall

La **Recall** è definita come il rapporto tra i true positive e la somma tra true positive e false negative.

$$R = \frac{tp}{tp + fn}$$

Precision e recall



F1 Score

L’ **F1 Score** è un punteggio che misura l’accuratezza di un test, una volta noti i valori di Precision e Recall.

Esso viene calcolato come:

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R}$$

Confusion matrix

Calcolare la Confusion Matrix, la precision, la recall e l’F1 Score del modello

Confusion matrix

file: 24_ngrams_confusion.py

Generazione

Al contrario della classificazione, i task di **generazione** hanno come obiettivo quello di completare un documento parziale.

Il testo viene completato inserendo sempre la parola più probabile, note le precedenti (...per il momento)

Generazione

Implementare la generazione con gli n-gram, data la prima parte di una recensione.

Se non esiste nessuna parola che completa l'ngram, oppure se il nuovo token è uguale al precedente, allora restituire una parola random scelta con `from random import choice`

Generazione

file: 25_ngrams_generation.py

Part of speech

La stessa parola, in contesti differenti, può appartenere a classi differenti: può essere un aggettivo, un soggetto, un verbo, etc.

Il **Part of Speech** associato ad una parola è la sua classe di appartenenza in quel contesto specifico.

Part of speech tagging

Il **Part of Speech Tagging** è un task di classificazione del testo.

In questo caso si considerano i casi in cui la stessa parola ha Part of Speech differenti.

Per addestrare un modello in questo modo avremmo bisogno, ovviamente, di un dataset molto più grande, che copra tutti i contesti possibili.

Named Entity Recognition

All’interno di un testo spesso si trovano **nomi** di

- persone
- luoghi
- aziende
- etc.

Named Entity Recognition

Il task del **Named Entity Recognition (NER)** è un task di classificazione che si tratta allo stesso modo del POS

Significato delle parole

In molti casi, si potrebbero utilizzare parole differenti (magari sinonimi), e la frase rimarrebbe identica dal punto di vista semantico.

Con le tecniche utilizzate finora ciò non risulta vero perché le parole sono state considerate come oggetti disgiunti.

Embeddings

Invece che rappresentare una parola con un identificativo univoco, vorremmo che essa fosse rappresentata da un punto in uno spazio *n*-dimensionale, in modo da poter anche misurare la distanza tra le parole.

In questo modo, un determinato punto nello spazio, rappresenterebbe un concetto, e non solamente una parola.

Questo tipo di rappresentazione è chiamato **Embeddings**.

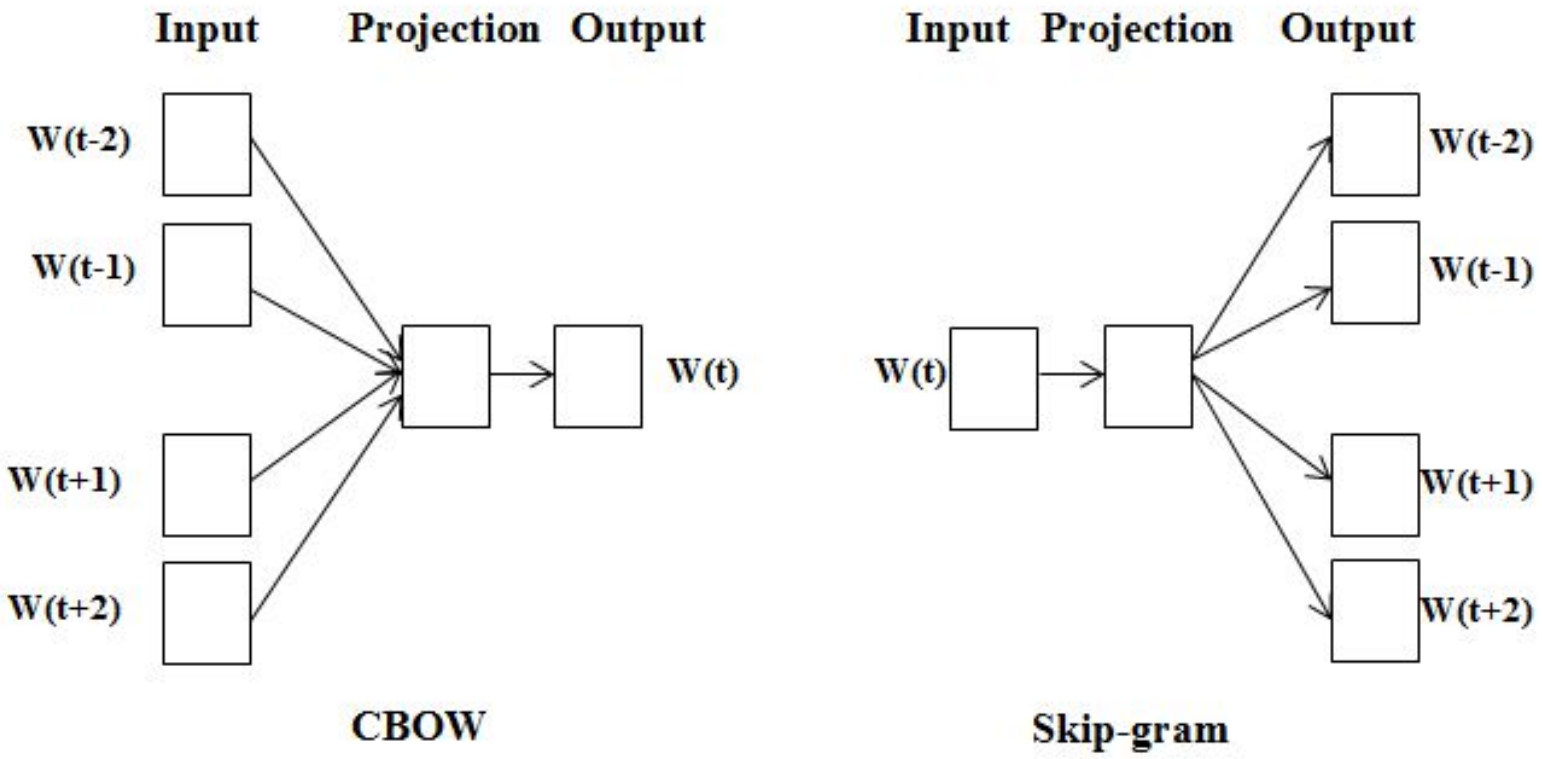
Word2Vec

Il punto di svolta negli embeddings è stato **Word2Vec**, sviluppato in Google nel 2013.

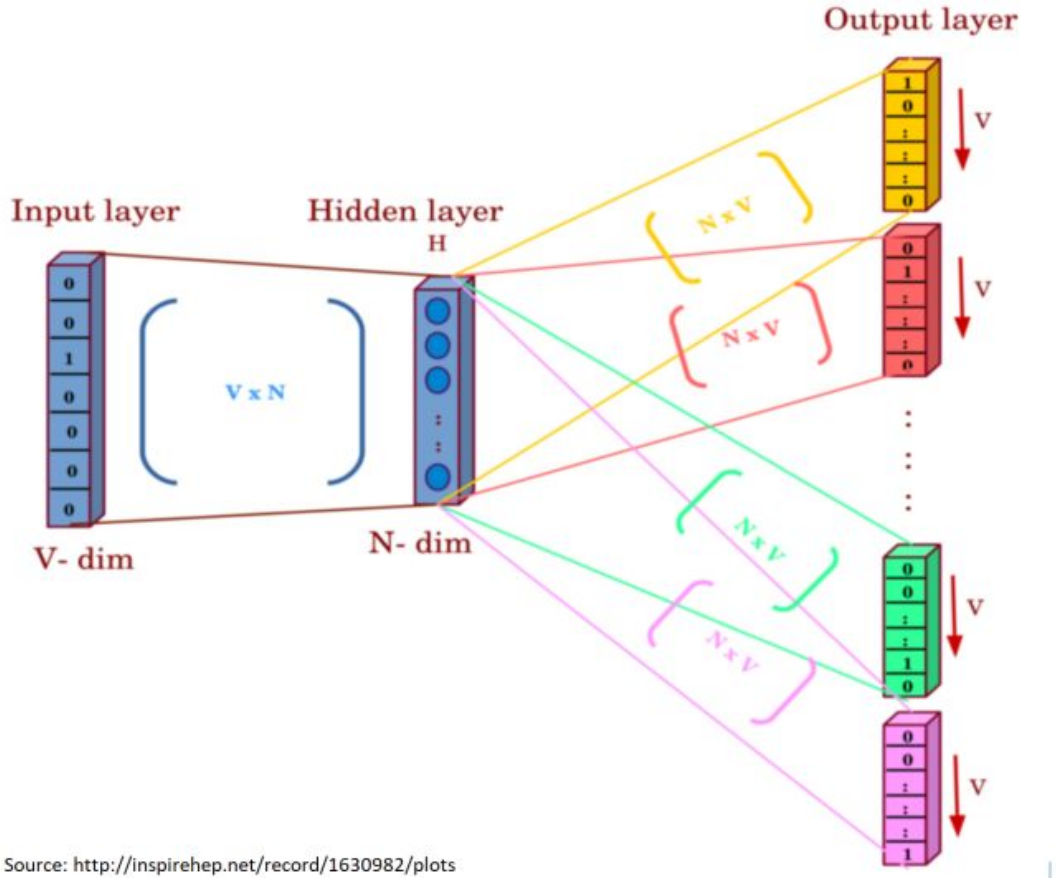
Esso crea rappresentazioni vettoriali delle parole in base al contesto.

CBOW & Skip-grams

Word2Vec si basa su due approcci diversi: CBOW e Skip-grams



Skip-Gram (architettura)



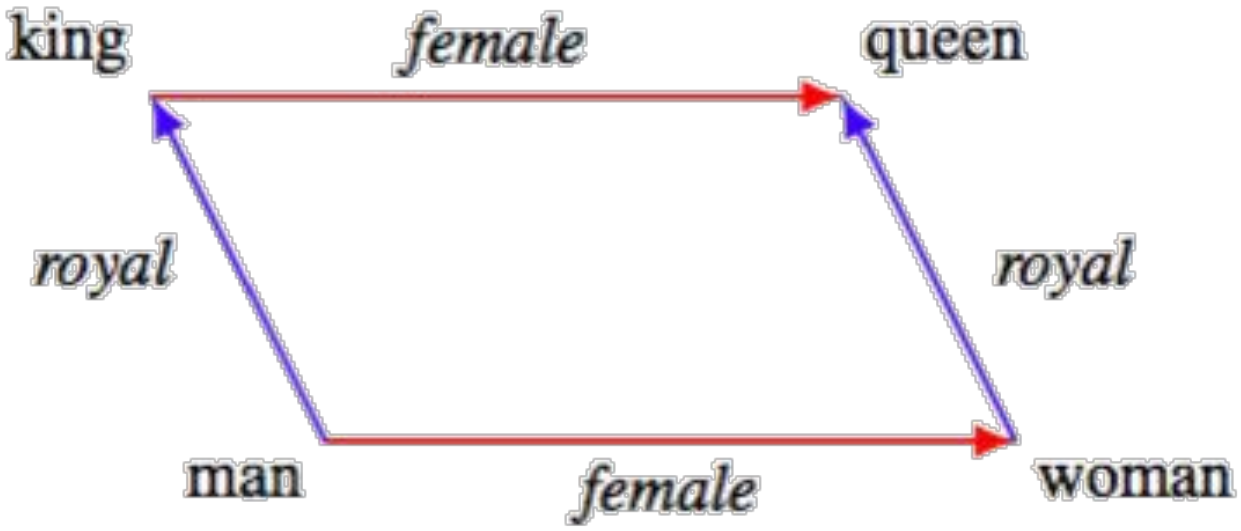
Skip-Gram Model

Source: http://inspirehep.net/record/1630982/plots

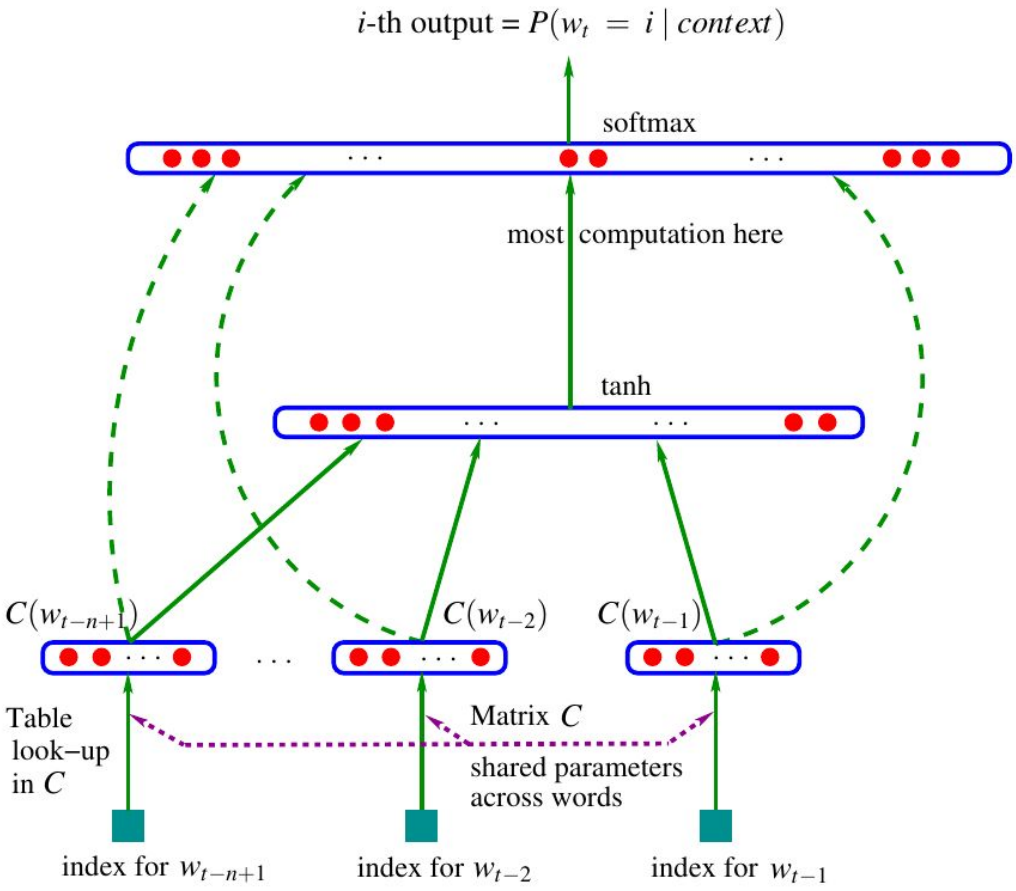
Risultati notevoli

Word2Vec è stato il primo a poter fare cose del genere

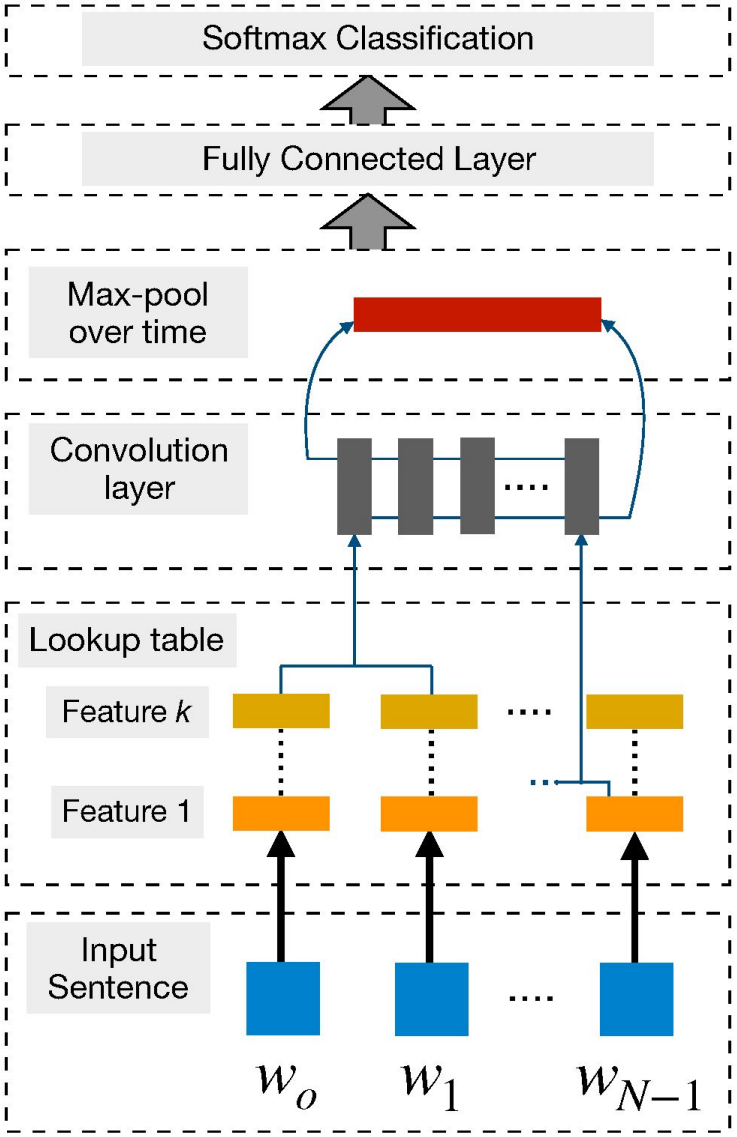
King - Man + Woman = Queen



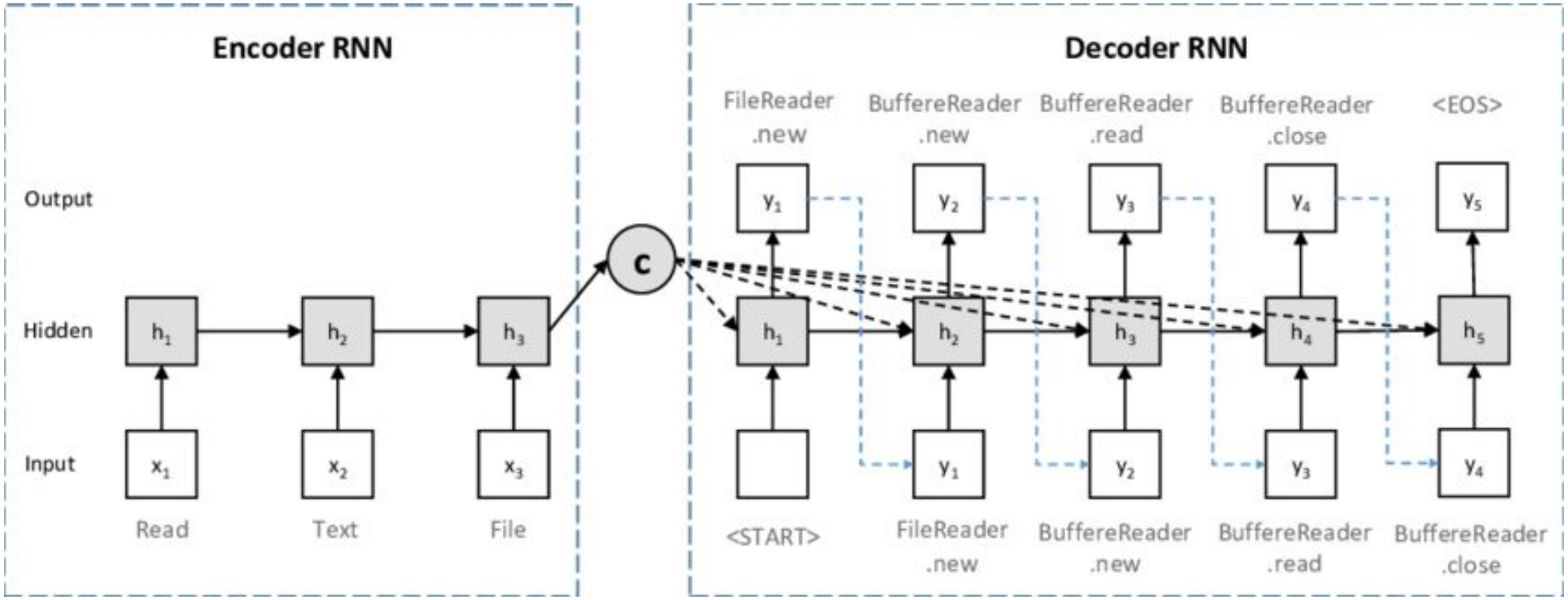
NN Language model



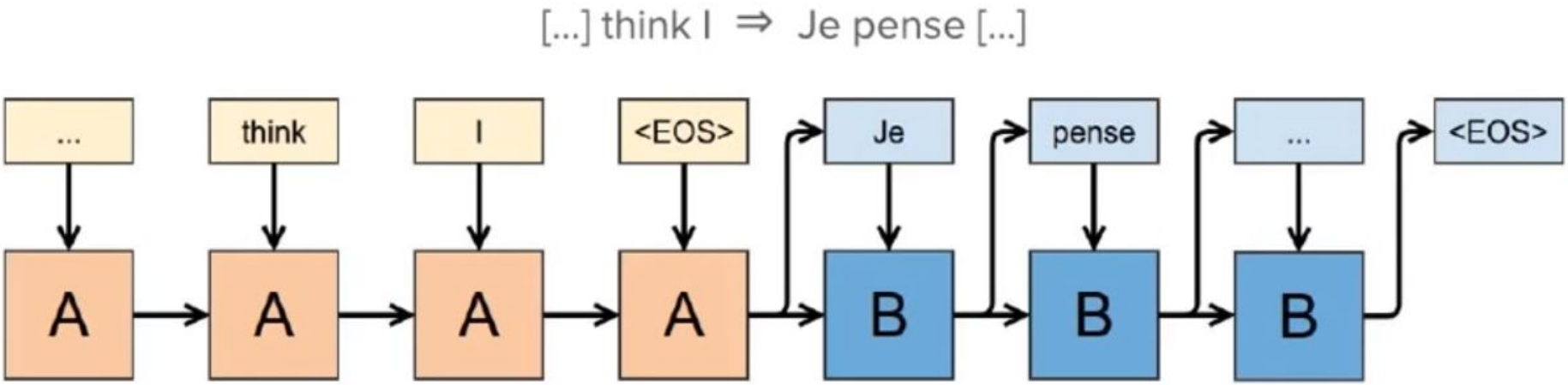
CNN-LM


Softmax Classification
Fully Connected Layer
Max-pool over time
Convolution layer
Lookup table
Feature k
Feature 1
Input Sentence
w_o
w_1
w_{N-1}

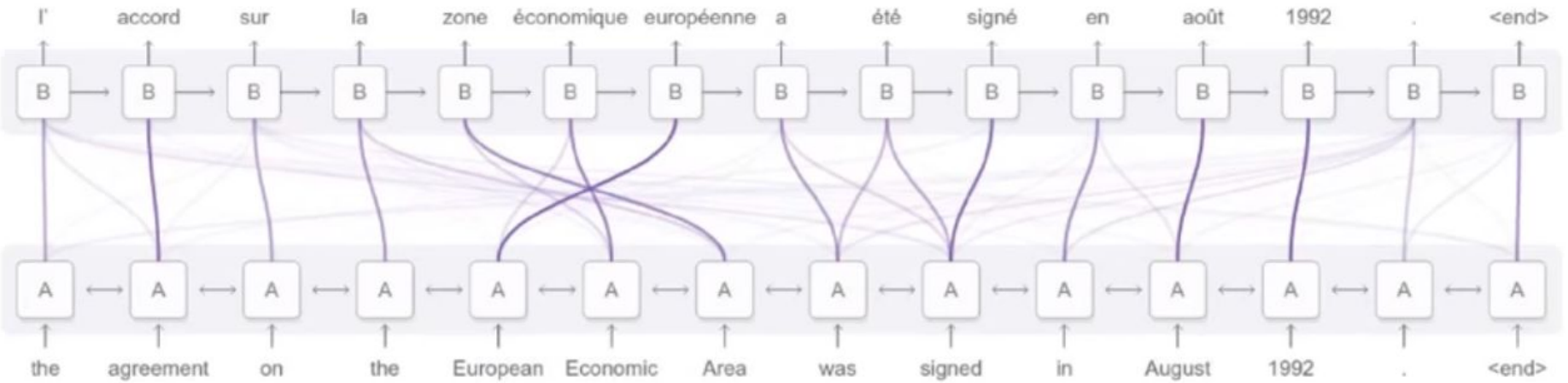
RNN Encoder-Decoder



Dalla fine



Inversioni


l' accord sur la zone économique européenne a été signé en août 1992 . <end>
B B B B B B B B B B B B B B B B
A A A A A A A A A A A A A A A A
the agreement on the European Economic Area was signed in August 1992 . <end>

Encoder-Decoder (problemi)

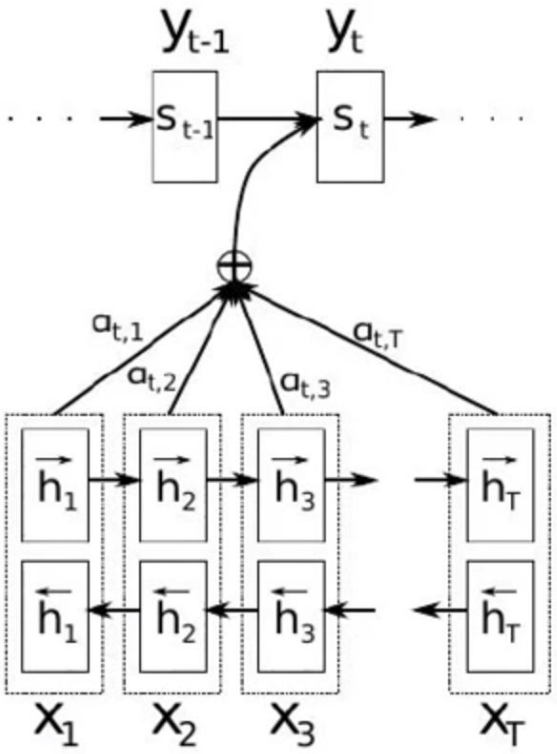
L’encoder deve riuscire a comprimere tutta l’informazione in un singolo vettore di dimensione fissata, indipendentemente dalla lunghezza della sequenza

Attention

Idea

Quando leggiamo una lunga frase, non cerchiamo di ricordare tutte le parole utilizzate, ma ci focalizziamo solamente su quelle “essenziali”

Attention

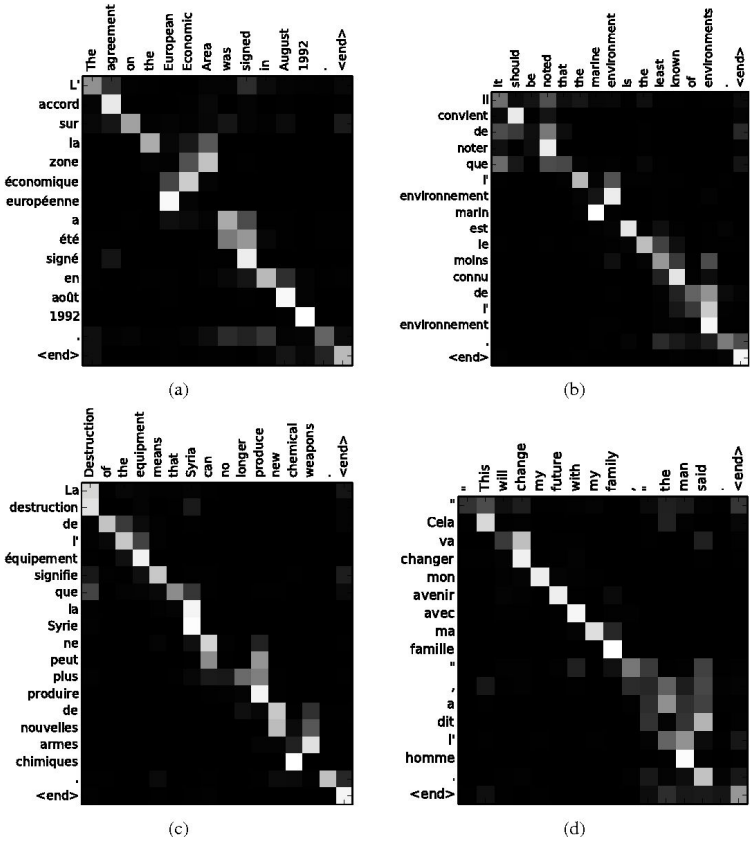


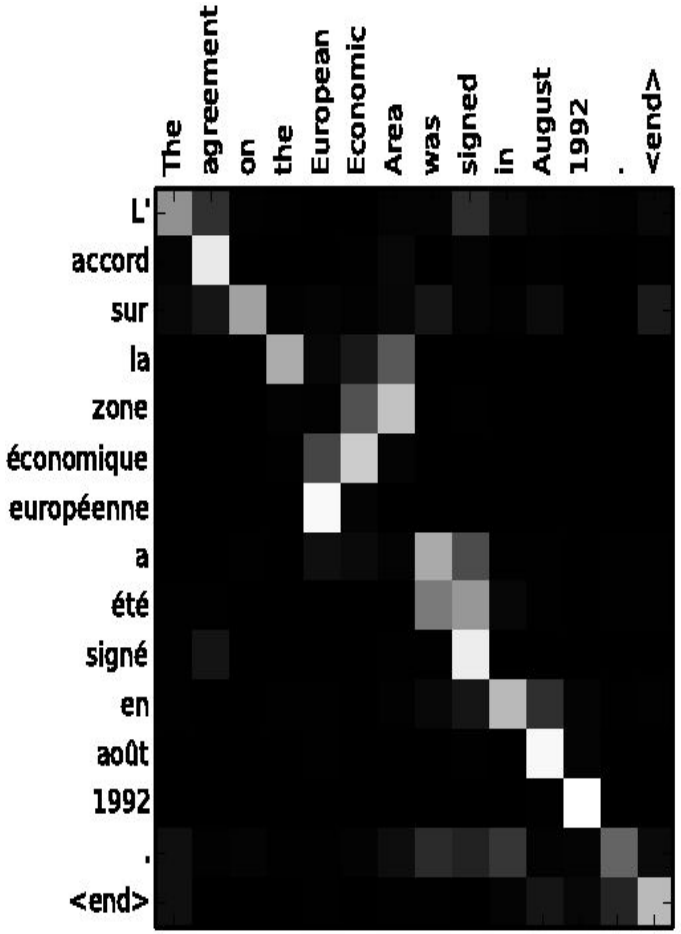
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)}, \qquad e_{ij} = a(s_{i-1}, h_j)$$

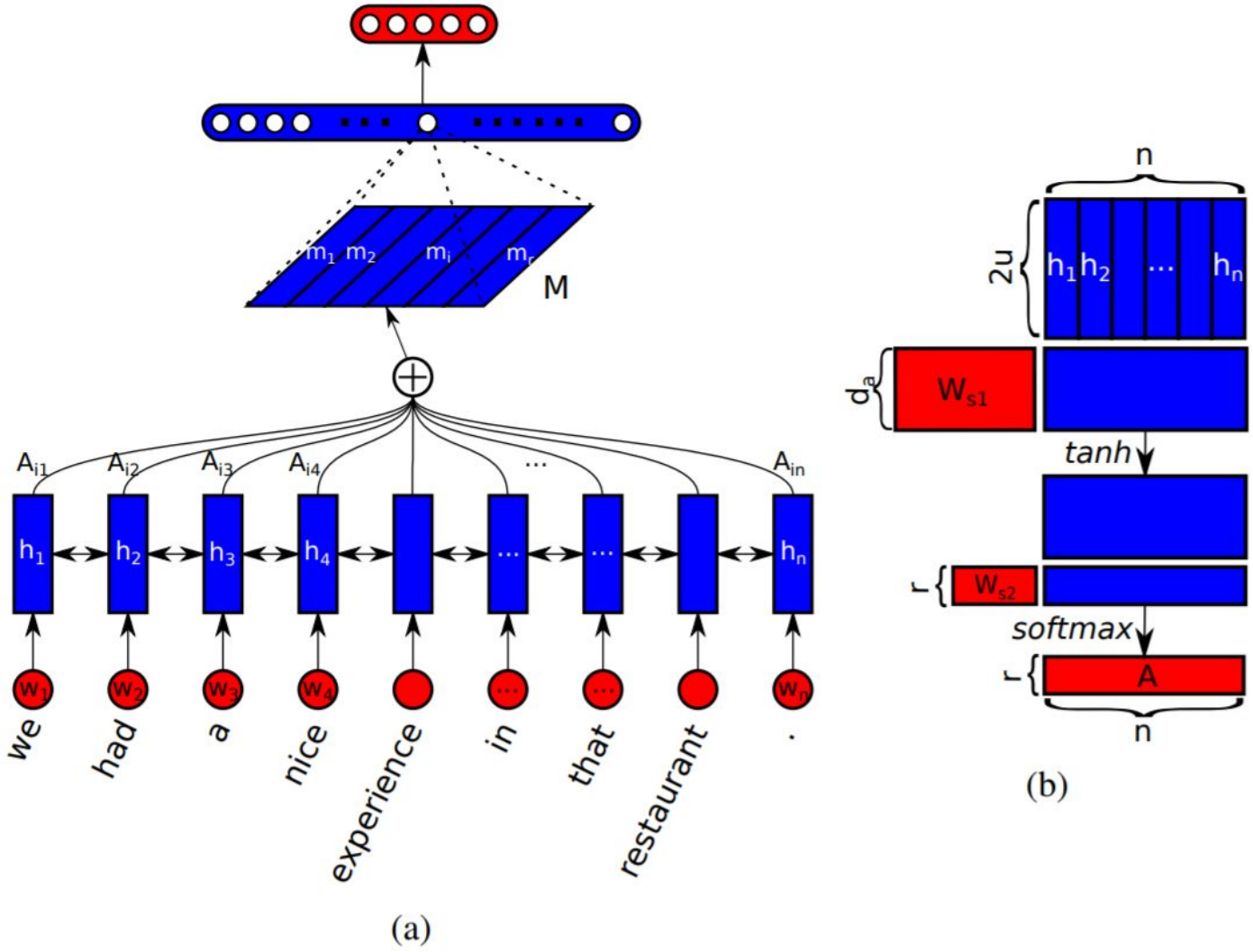
$$a(s_{i-1}, h_j) = v_a^\top \tanh\left(W_a s_{i-1} + U_a h_j\right),$$

Attention (visualizzata)

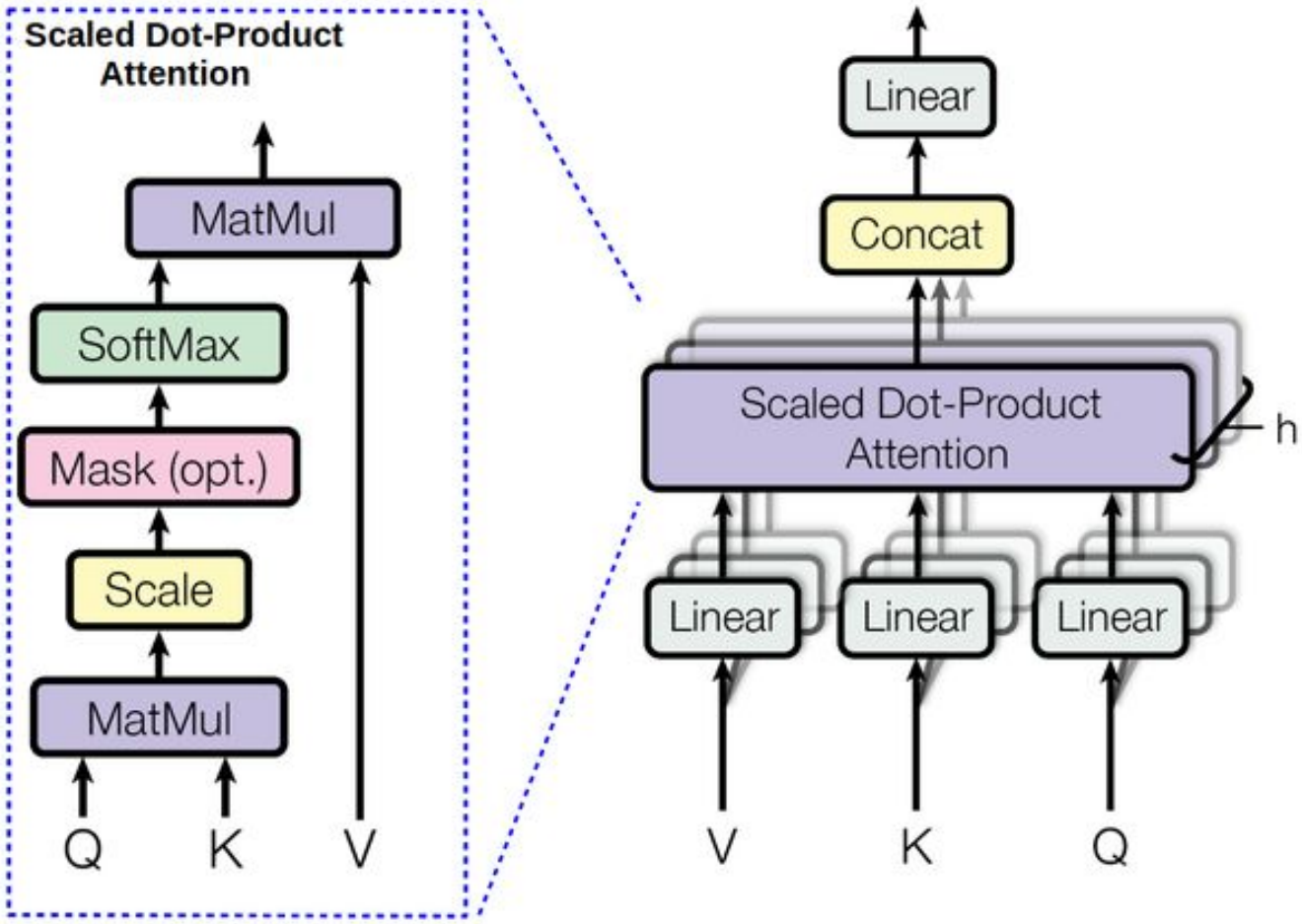




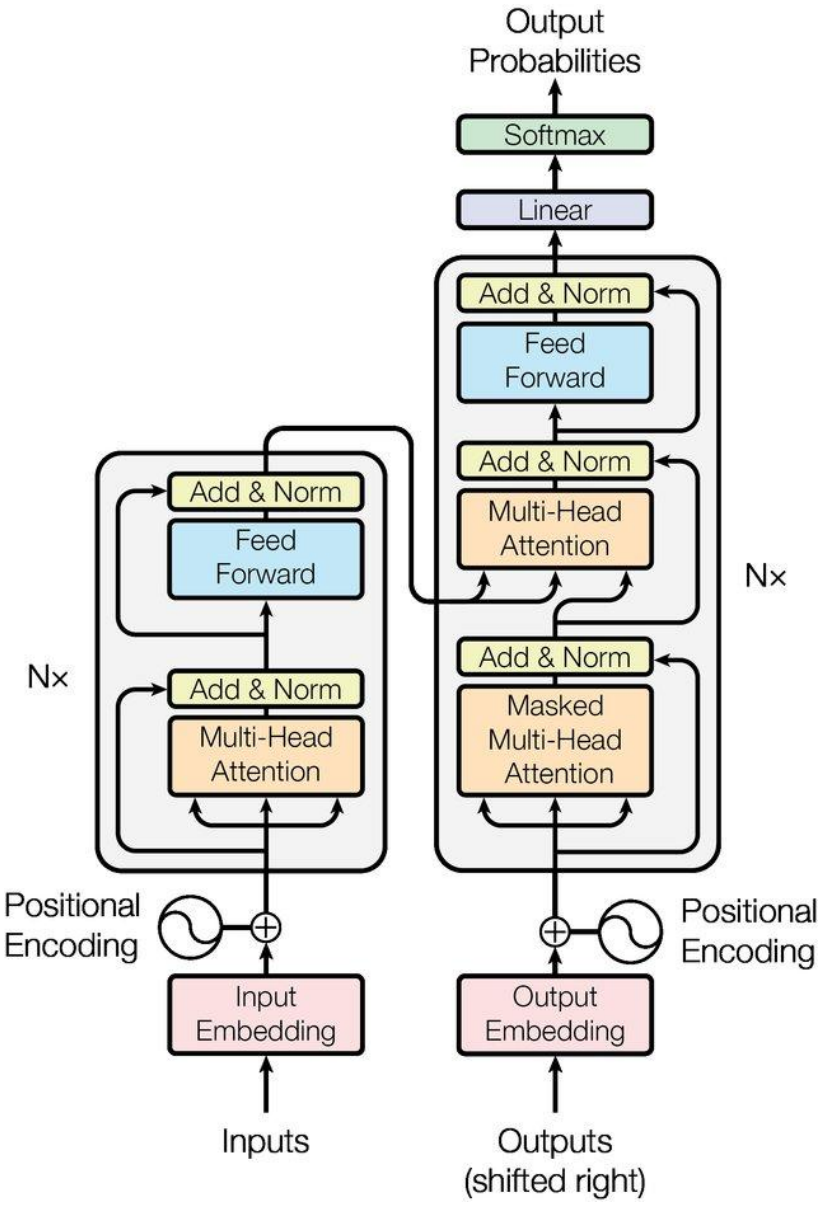
Self-attention



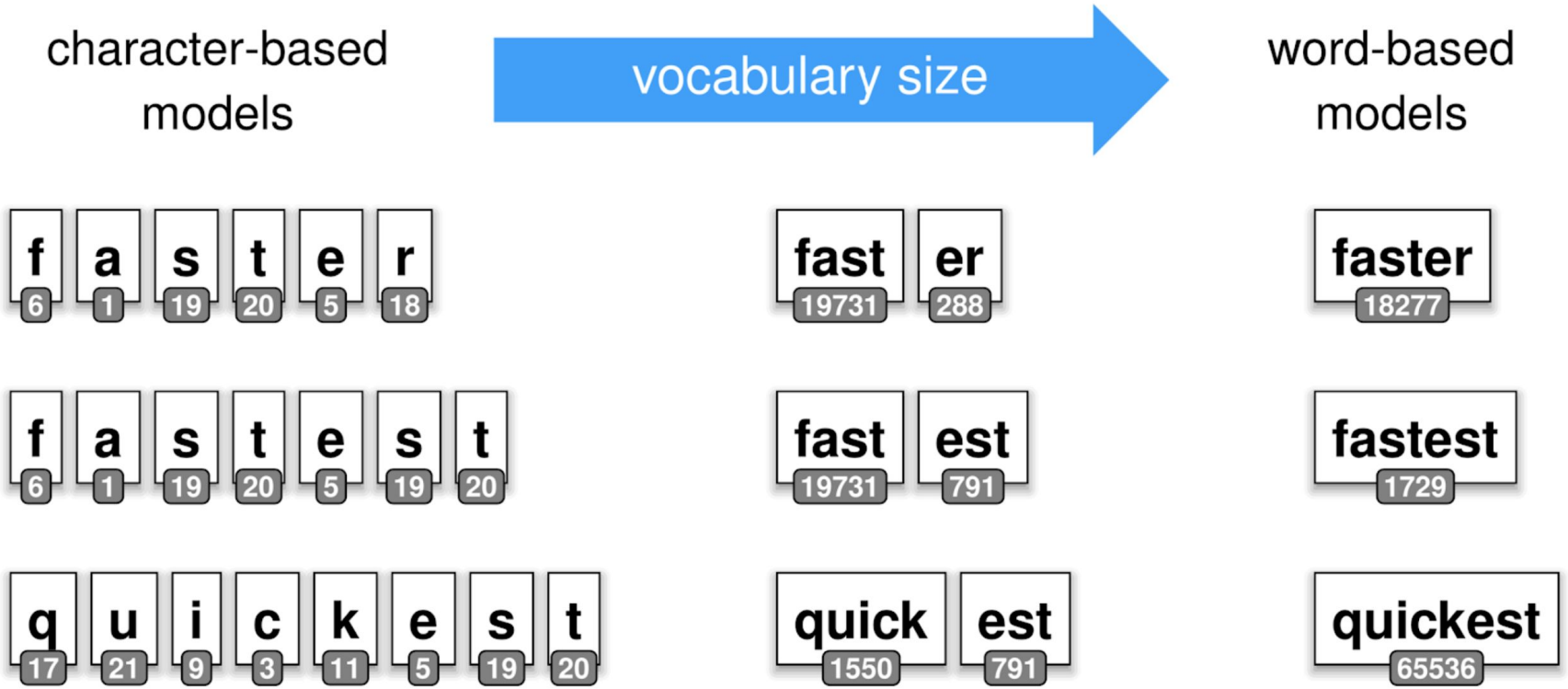
Multi-head attention



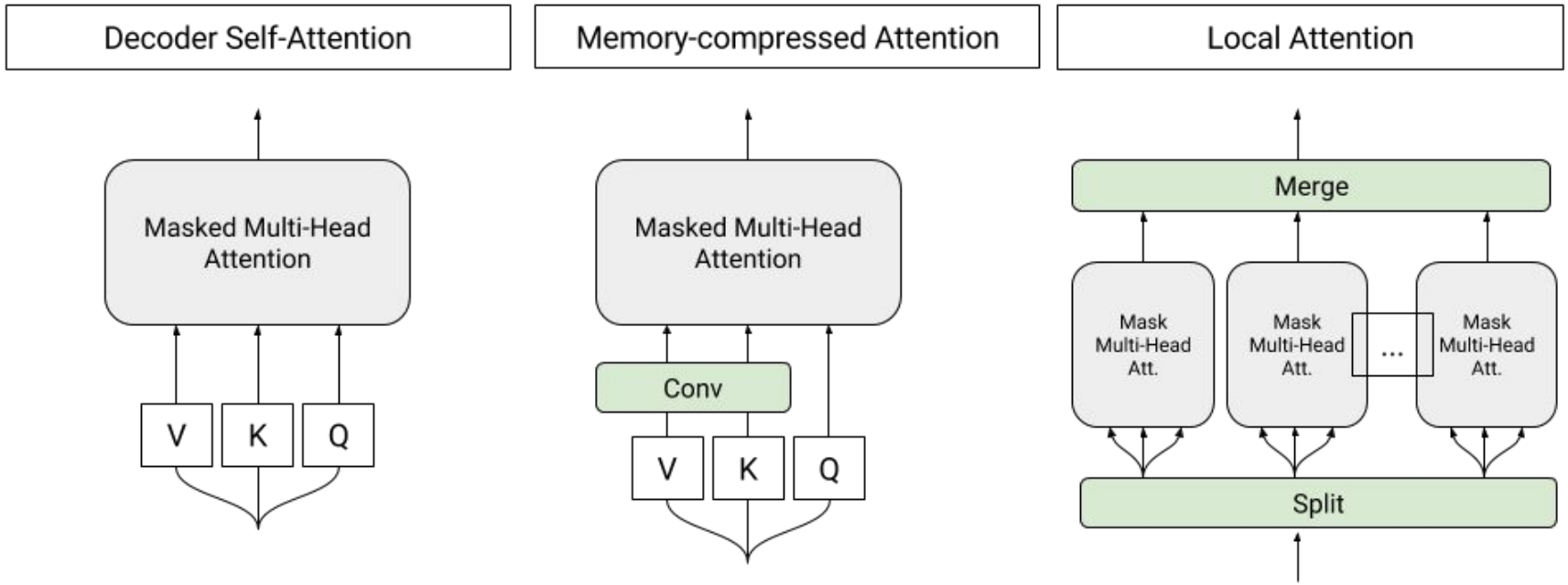
Transformer



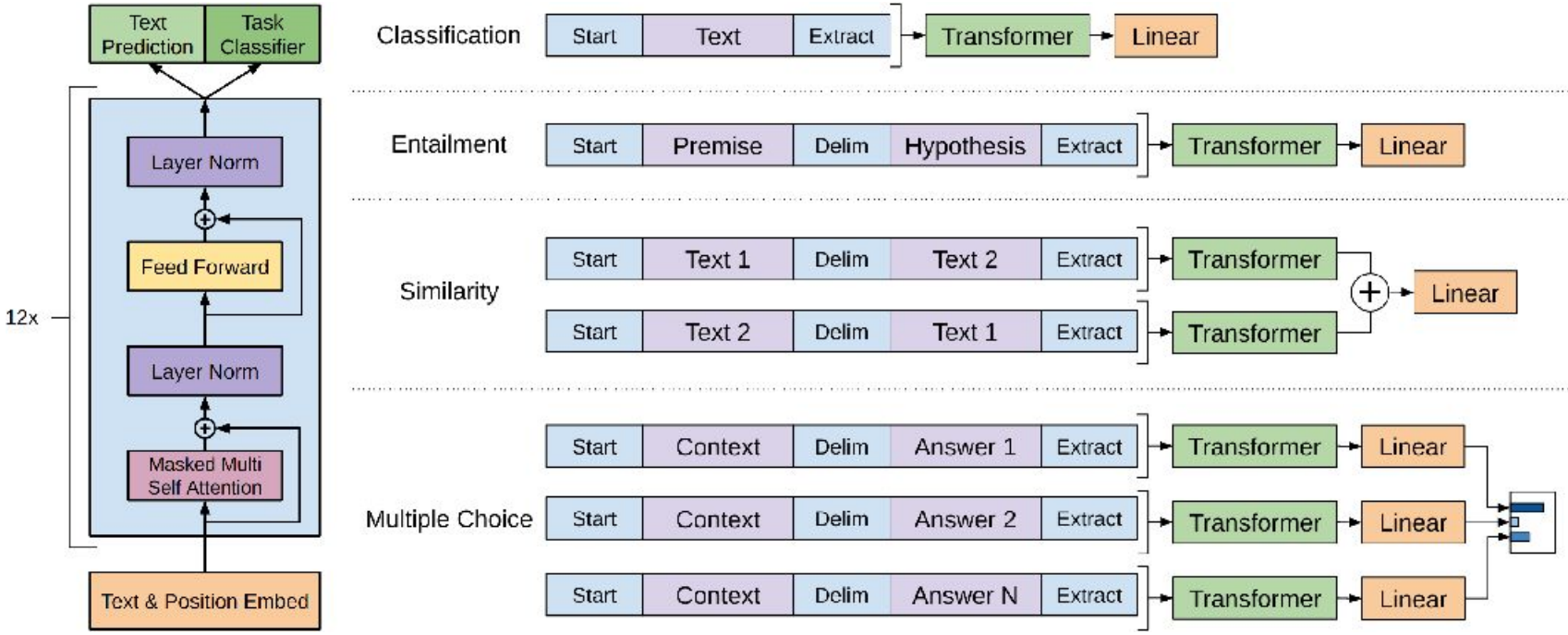
Subword tokenization



Riassumendo wikipedia

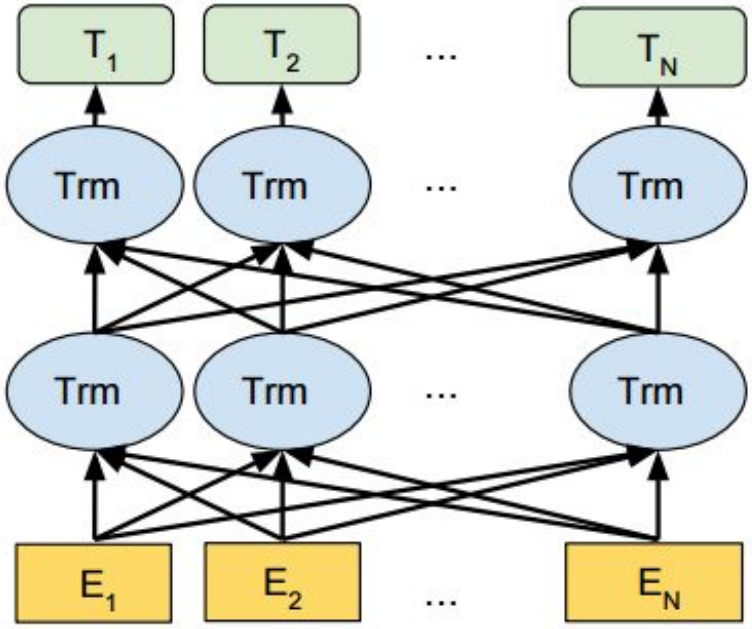


GPT-1

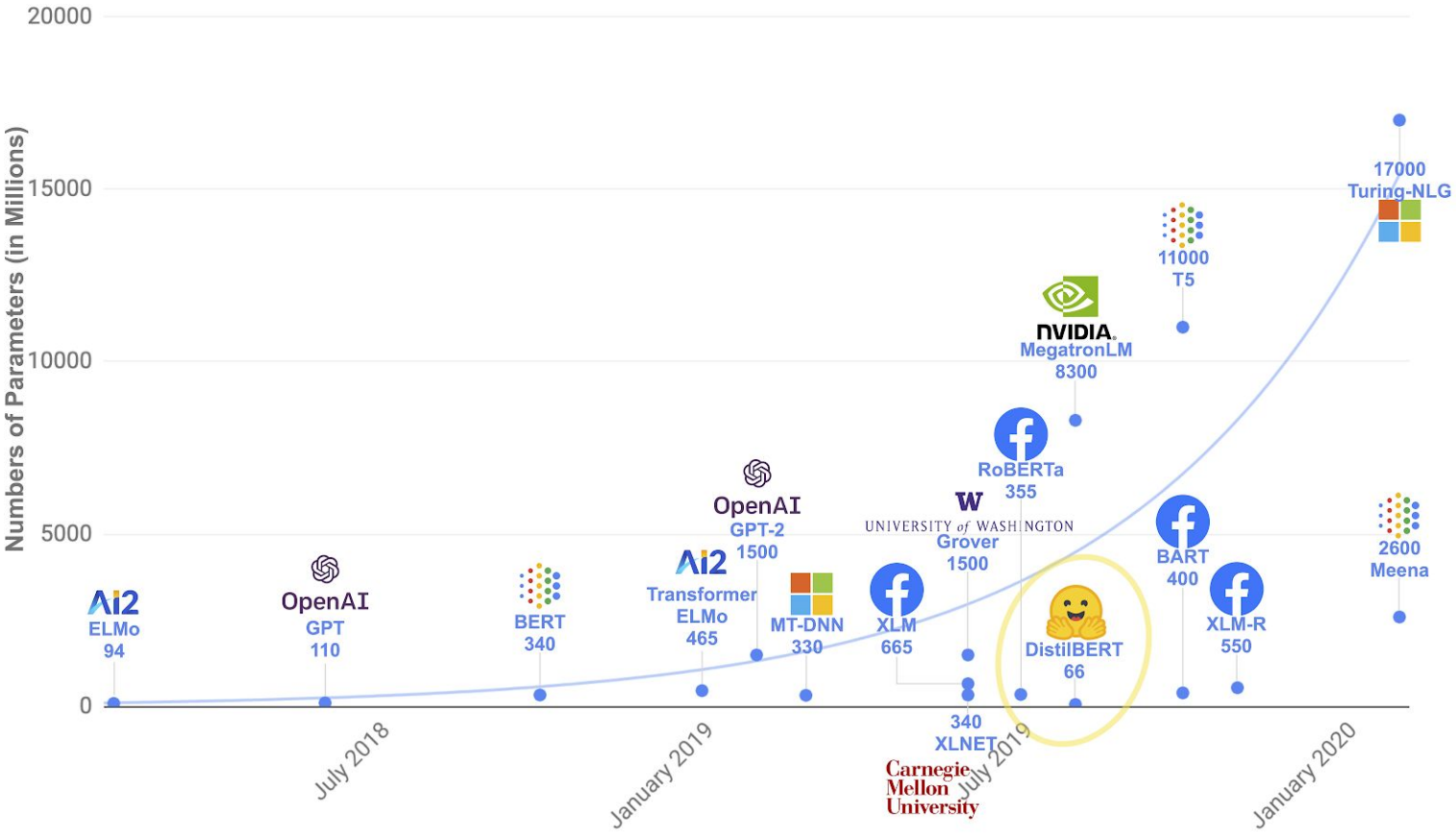

Text Prediction
Task Classifier
Layer Norm
Feed Forward
Layer Norm
Masked Multi Self Attention
12x
Text & Position Embed
Classification
Start
Text
Extract
Transformer
Linear
Entailment
Start
Premise
Delim
Hypothesis
Extract
Transformer
Linear
Similarity
Start
Text 1
Delim
Text 2
Extract
Transformer
Start
Text 2
Delim
Text 1
Extract
Transformer
Linear
Multiple Choice
Start
Context
Delim
Answer 1
Extract
Transformer
Linear
Start
Context
Delim
Answer 2
Extract
Transformer
Linear
Start
Context
Delim
Answer N
Extract
Transformer
Linear

GPT-2

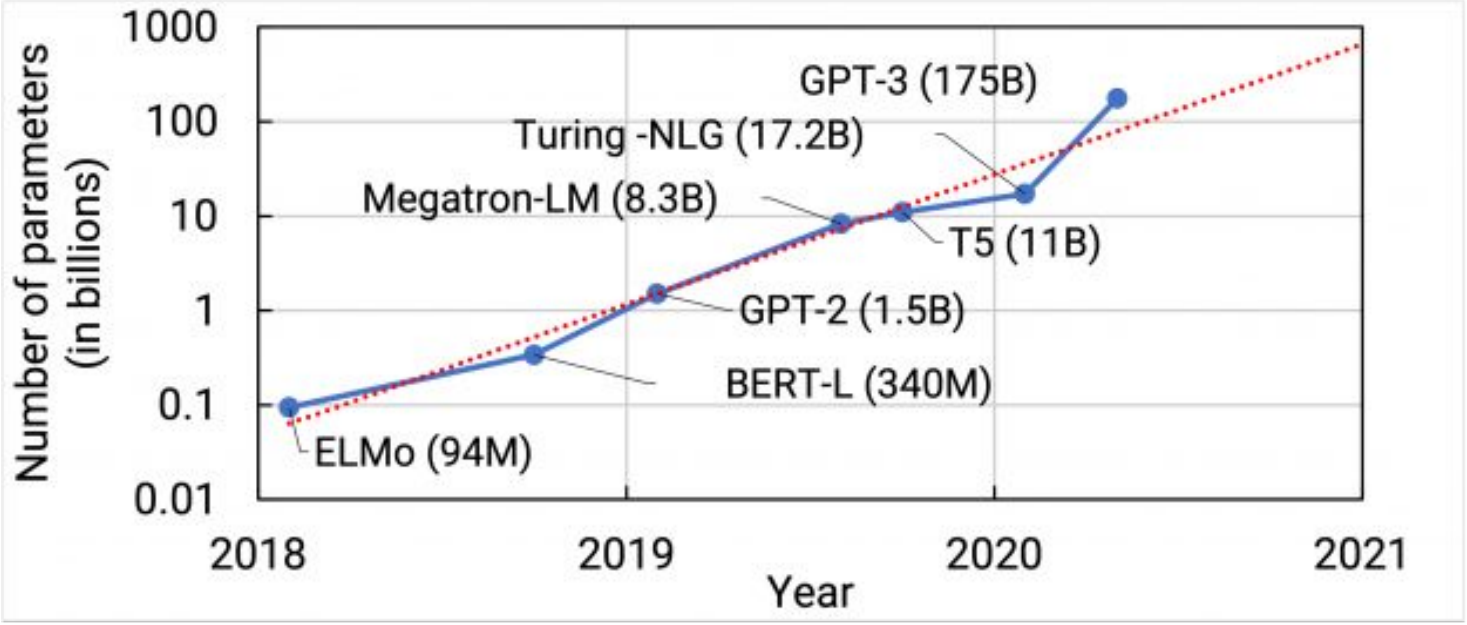
Google BERT



Bigger is better?



Bigger is better?



Il train di GPT-3 è costato circa 12 milioni di dollari (solo di crediti cloud)