Neural Language Models

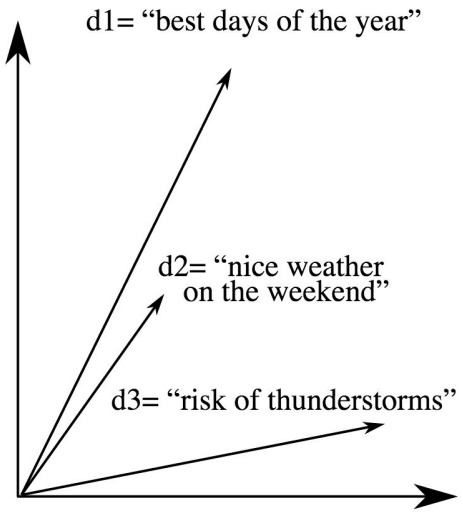
Text Analytics - Andrea Esuli

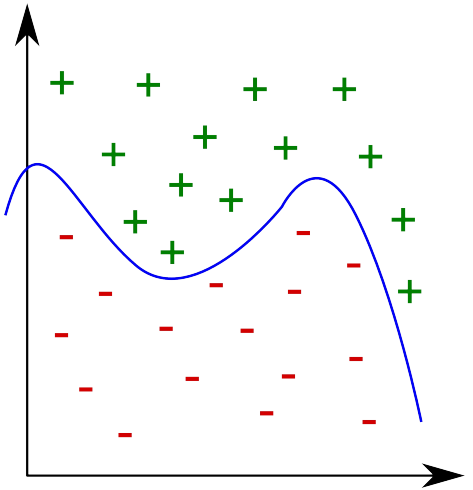
Vector Space Model

The *Vector Space Model* (VSM) is a typical machine-processable representation adopted for text.

Each vector positions a document into an *n*-dimensional space, on which learning algorithms operate to build their models

$$v(d_1) = [w_1, w_2 \ldots \ldots, w_{n-1}, w_n]$$





Vector Space Model

After text processing, tokenization... a document is usually represented as vector in $R^{|F|}$, where F is the set of all the distinct *features* observed in documents.

Each feature is mapped to a distinct dimension in $R^{|F|}$ using a *one-hot* vector:

$$
\begin{aligned}
v(\text{'played'}) &= [1, 0, 0, \ldots, 0, 0, \ldots, 0, 0, 0] \\
v(\text{'game'}) &= [0, 1, 0, \ldots, 0, 0, \ldots, 0, 0, 0] \\
v(\text{'match'}) &= [0, 0, 1, \ldots, 0, 0, \ldots, 0, 0, 0] \\
&\vdots \\
v(\text{'trumpet'}) &= [0, 0, 0, \ldots, 0, 1, \ldots, 0, 0, 0] \\
&\vdots \\
v(\text{'bwoah'}) &= [0, 0, 0, \ldots, 0, 0, \ldots, 0, 0, 1]
\end{aligned}
$$

Vector Space Model

A document is represented as the weighted sum of its features vectors:

$$v(d) = \sum_{f \in d} w_{fd} v(f)$$

For example:

d = 'you played a good game'

$v(d)$ = [0,$w_{played,d}$,$w_{game,d}$, 0,… …0, $w_{good,d}$, 0… …0, 0]

The resulting document vectors are sparse:

$$|\{i|v_i(d) \neq 0\}| \ll n$$

Sparse representations

d_1 = 'you played a game'
d_2 = 'you played a match'
d_3 = 'you played a trumpet'

$v(d_1)$ = [0, $w_{played,d1}$, $w_{game,d1}$, 0 , 0, … , 0, 0 , 0]
$v(d_2)$ = [0, $w_{played,d2}$, 0 , $w_{match,d2}$, 0, … , 0, 0 , 0]
$v(d_3)$ = [0, $w_{played,d3}$, 0 , 0 , 0, … , 0, $w_{trumpet,d3}$, 0]

Semantic similarity between features (game~match) is not captured:

$$sim(v(d_1), v(d_2)) \sim sim(v(d_1), v(d_3)) \sim sim(v(d_2), v(d_3))$$

Modeling word similarity

How do we model that *game* and *match* are related terms and *trumpet* is not?

Using linguistic resources: it requires a lot of human work to build them.

Observation: co-occurring words are semantically related.

Pisa	*is a*	*province*	*of*	*Tuscany*
Red	*is a*	*color*	*of the*	*rainbow*
Wheels	*are a*	*component*	*of the*	*bicycle*
**Red*	*is a*	*province*	*of the*	*bicycle*

We can exploit this propriety of language, e.g., following the *distributional hypothesis*.

Distributional hypothesis

“You shall know a word by the company it keeps!” Firth (1957)

Distributional hypothesis: the meaning of a word is determined by the contexts in which it is used.

*Yesterday we had **bwoah** at the restaurant.*
*I remember my mother cooking me **bwoah** for lunch.*
*I don't like **bwoah**, it's too sweet for my taste.*
*I like to dunk a piece **bwoah** in my morning coffee.*

Dense representations

We can learn a projection of feature vectors *v(f)* into a *low dimensional* space R^k, $k \ll |F|$, of *continuous space word representations* (a.k.a., embeddings!).

$$Embed: R^{|F|} \rightarrow R^k$$

$$Embed(v(f)) = e(f)$$

We force features to share dimensions on a reduced dense space

↓

Let's group/align them by their syntactic/semantic similarities!

Word-Context matrix

A word-context (or word-word) matrix is a $|F| \cdot |F|$ matrix X that **counts** the frequencies of co-occurrence of words in a collection of contexts (i.e, text spans of a given length).

You cook the cake twenty minutes in the oven at 220 C.
I eat my steak rare.
I'll throw the steak if you cook it too much.
The engine broke due to stress.
I broke a tire hitting a curb, I changed the tire.

$\text{Context}_{-2,+2}$('cake') = {['cook','the','twenty', 'minutes']}
$\text{Context}_{-2,+2}$('tire') = {['broke','a','hitting', 'a'], ['changed', 'the']}

Word-Context matrix

		Context words						
		…	cook	eat	…	changed	broke	…
Words	cake	…	10	20	…	0	0	…
	steak	…	12	22	…	0	0	…
	bwoah	…	7	10	…	0	0	…
	engine	…	0	0	…	3	10	…
	tire	…	0	0	…	10	1	…
	…	…	…	…	…	…	…	…

$Words \equiv Context\ words$

Rows of X capture similarity yet X is still high dimensional and sparse.

SVD

Singular Value Decomposition is a decomposition method of a matrix X of size $m \cdot n$ into three matrices $U\Sigma V^*$, where:

U is an orthonormal matrix of size $m \cdot n$

Σ is a diagonal matrix of size $n \cdot n$

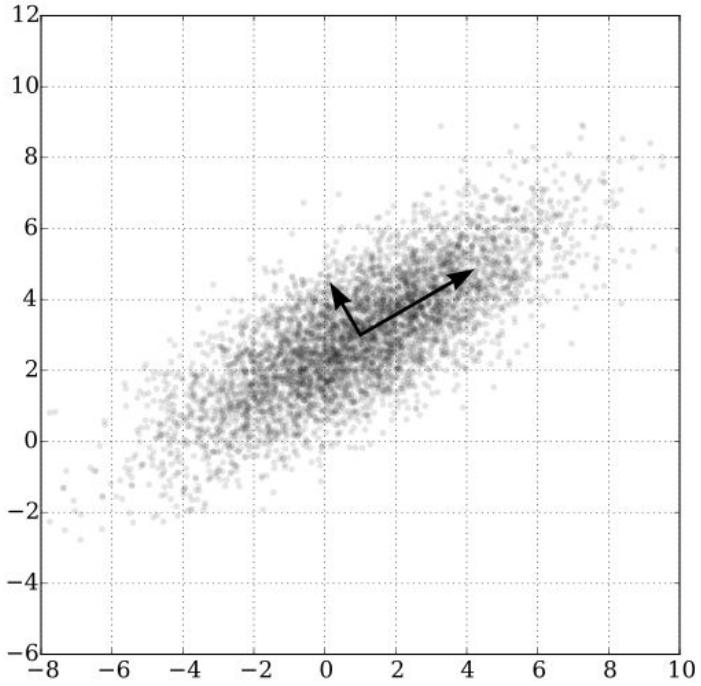
V is an orthonormal matrix of size $n \cdot n$, V^* is its conjugate transpose

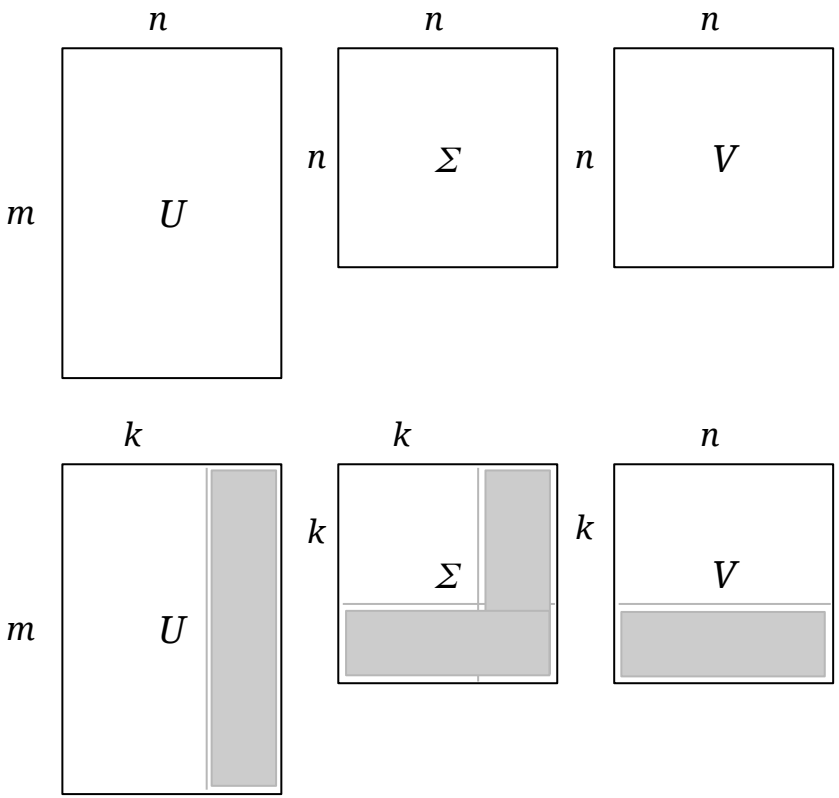
$\sigma_1, \sigma_2 \dots \sigma_n$ values of Σ are the singular values of X

Keeping the top k values is a least-square approximation of X

Rows of U_k of size $m \cdot k$ are the dense representations of the features

SVD

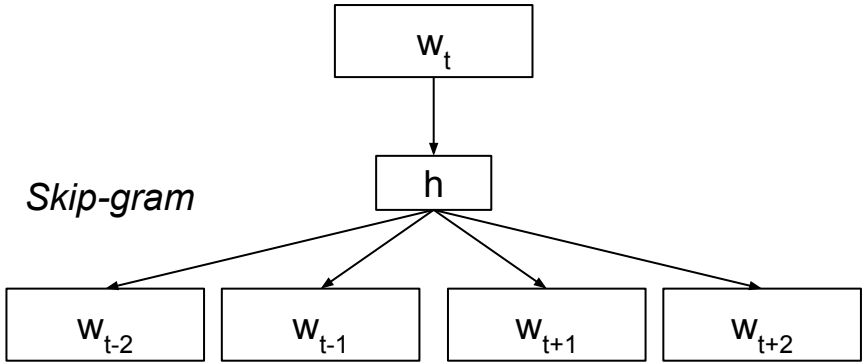


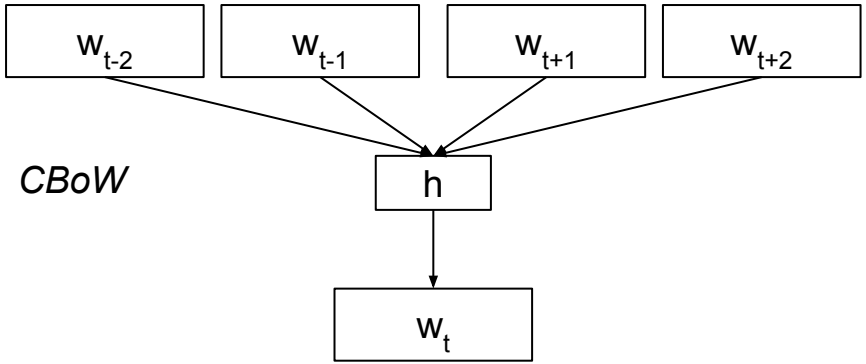


Word2Vec

Skip-gram and CBoW models of Word2Vec define tasks of **predicting** a context from a word (Skip-gram) or a word from its context (CBoW).

They are both implemented as a two-layers linear neural network in which input and output words one-hot representations which are encoded/decoded into/from a dense representation of smaller dimensionality.





Word2Vec

Embeddings are a by-product of the word prediction task.

Even though it is a prediction tasks the network can be trained on any text, no need for human-labeled data!

Usual context window size is two words before, two words after.

Features can be also multi-word expressions.

Longer windows capture more semantic, less syntax.

A typical size for h is 200~300.

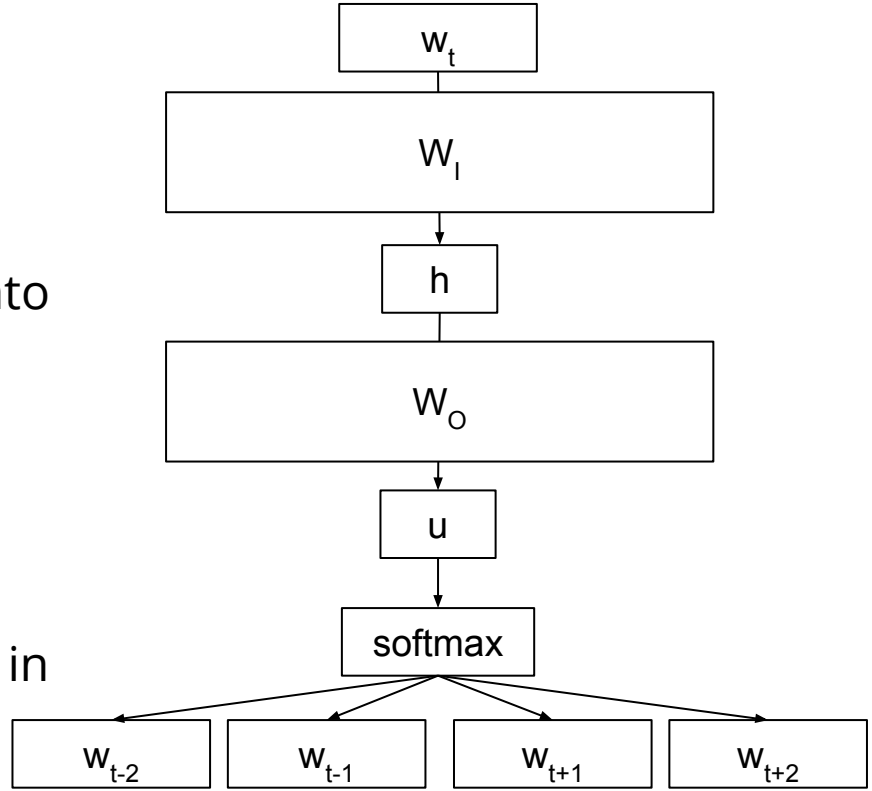
Skip-gram

w vectors are high dimensional, $|F|$

h is low dimensional (it is the size of the embedding space)

W_I matrix is $|F| \cdot |h|$. It encodes a word into a hidden representation.
Each row of W_I defines the embedding of the a word.

W_O matrix is $|h| \cdot |F|$. It defines the embeddings of words when they appears in contexts.



Skip-gram

$h = w_t W_I \leftarrow$ h is the embedding of word w_t

$u = h\ W_O \leftarrow$ u_i is the similarity of h with context embedding of w_i in W_O

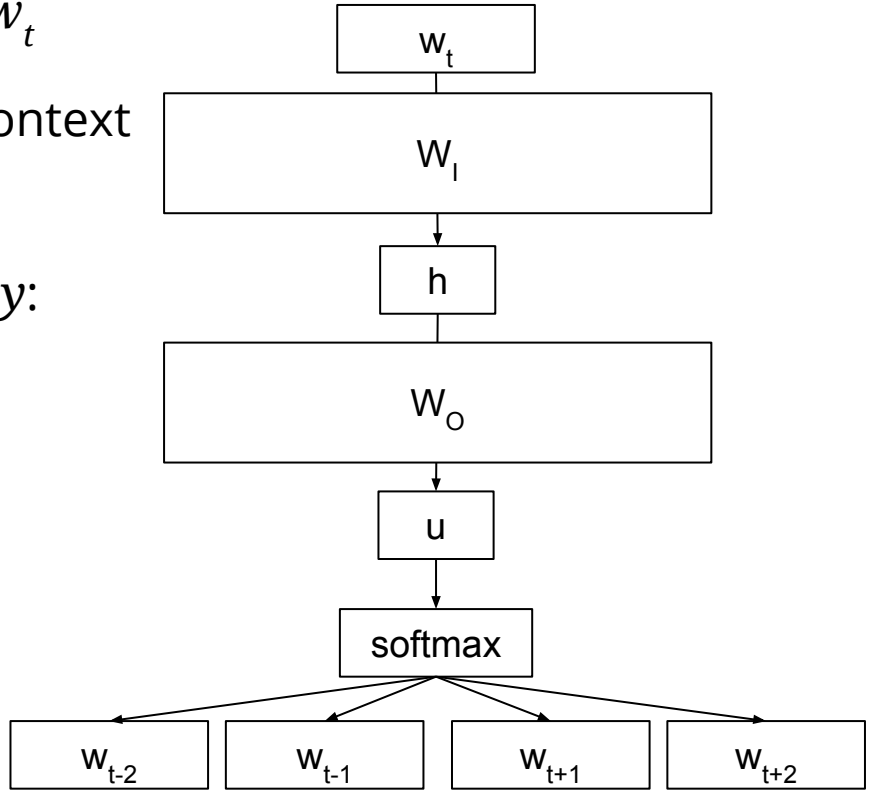
Softmax converts u to a prob distribution y:

$$y_i = exp(u_i) / \sum_{j \in F} exp(u_j)$$

Loss:

$$- log\ p(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} \mid w_t) =$$
$$= - log \prod_{c \in C} exp(u_c) / \sum_{j \in F} exp(u_j) =$$
$$= - \sum_{c \in C} exp(u_c) + C\ log \sum_{j \in F} exp(u_j)$$

i.e., maximize probability of context, minimize probability of the rest



Negative sampling

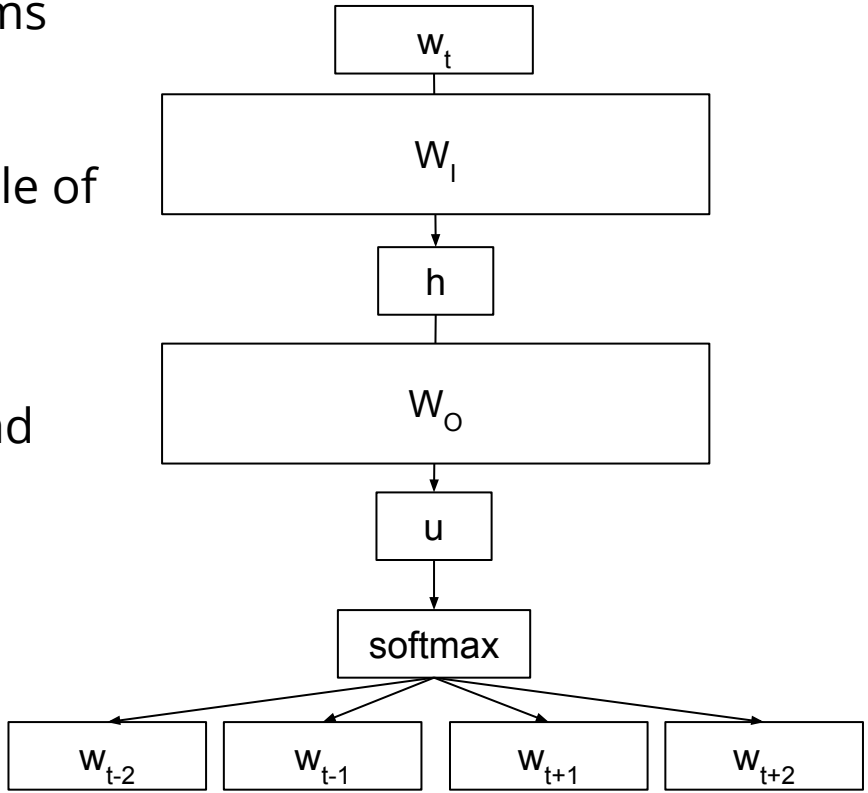
The $log \sum_{j \in F} exp(u_j)$ factor has a lots of terms and it is costly to compute.

Solution: compute it only on a small sample of negative examples, i.e.,

$$log \sum_{j \in E} exp(u_j)$$

where words in E are just a few (e.g., 5) and they are sampled using a biased unigram distribution computed on training data:

$$p(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j \in F | f(w_j)^{3/4}}}$$



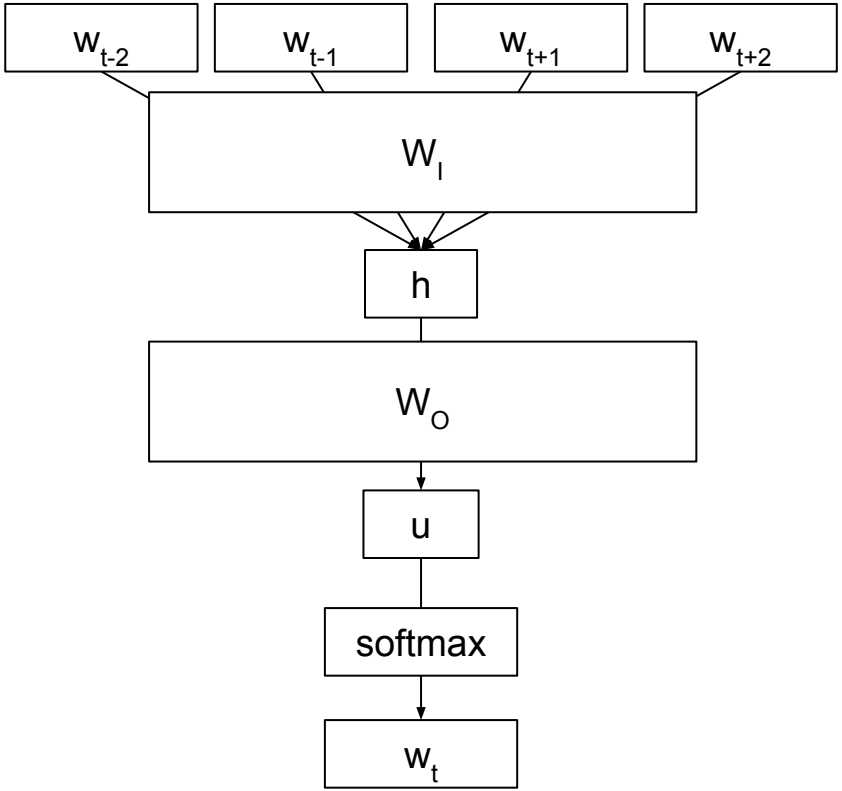
CBoW

CBoW stands for Continuous Bag of Word.

It's a mirror formulation of the skip-gram model, as context words are used to predict a target word.

h is the average of the embedding for the input context words.

u_i is the similarity of h with the words embedding w_i in W_O



GloVe

GloVe: Global Vectors for Word Representation is a count-based model that factorizes the word-context matrix based on the observation that the ratio of conditional probabilities better captures the semantic relations between words.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

← eqn 8 of GloVe paper

Which one?

Levy and Goldberg proved that Word2Vec skip-gram with negative sampling (SGNS) implicitly computes a factorization of a variant of X.

Levy, Goldberg and Dagan ran an extensive comparison of SVD, CBoW, SGNS, GloVe.

- Results indicate no clear overall winner.
- Parameters play a relevant role in the outcome of each method.
- Both SVD and SGNS performed well on most tasks, never underperforming significantly.
- SGNS is suggested to be a good baseline, given its lower computational cost in time and memory.

Computing embeddings

The training cost of Word2Vec is linear in the size of the input.

The training algorithm works well in parallel, given the sparsity of words in contexts and the use of negative sampling. The probability of concurrent update of the same values by two processes is minimal → let's ignore it when it happens (a.k.a., asynchronous stochastic gradient descent).

Can be halted/restarted at any time.

The model can be updated with any data (concept drift/ domain adaptation).

Computing embeddings

Gensim provides an efficient and detailed implementation.

```
sentences = [['this','is','a','sentence'],
             ['this','is','another','sentence']]

from gensim.models import Word2Vec
model = Word2Vec(sentences)
```

This is a clean implementation of skip-grams using pytorch.

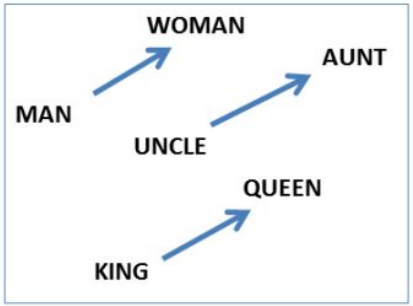
Which embeddings?

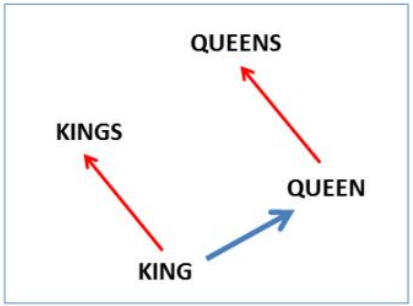
Both W_I and W_O define embeddings, which one to use?

- Usually just W_I is used.
- Average pairs of vectors from W_I and W_O into a single one.
- Append one embedding vector after the other, doubling the length.

Testing embeddings

Testing if embeddings capture syntactic/semantic properties.





Analogy test:

Paris	stands to	France	as	Rome	stands to	?
Writer	stands to	book	as	painter	stands to	?
Cat	stands to	cats	as	mouse	stands to	?

$$e('France') - e('Paris') + e('Rome') \sim e('Italy')$$

a : b = c : d

$$d = \arg\max_x \frac{(e(b) - e(a) + e(c))^T e(x)}{||e(b) - e(a) + e(c)||}$$

Precomputed embeddings

Google's embeddings trained on English news dataset (100 billion of words occurrences), plus other relevant resources.

Embeddings for German.

Embeddings for Italian.

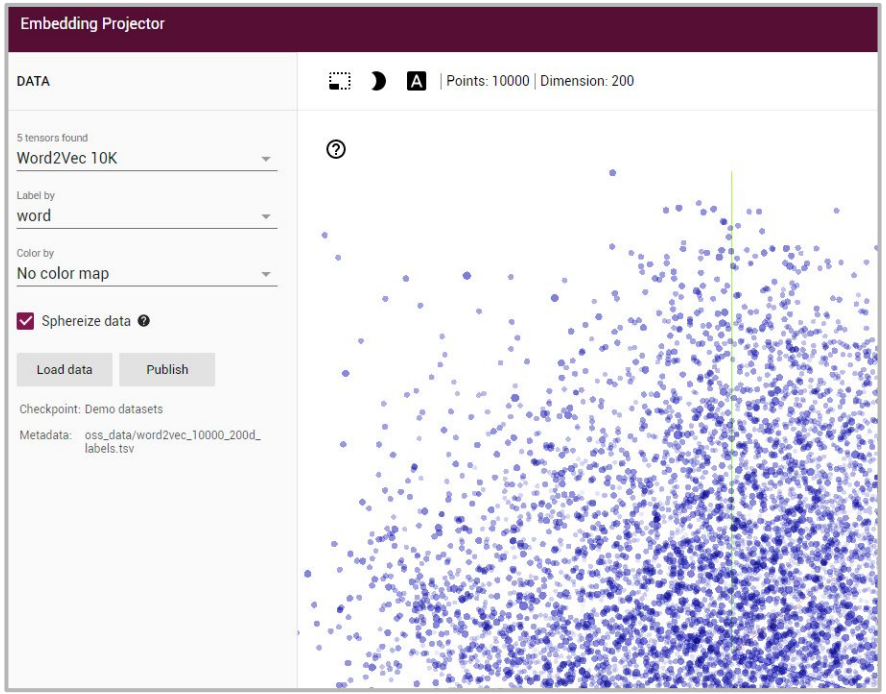
Embeddings for French.

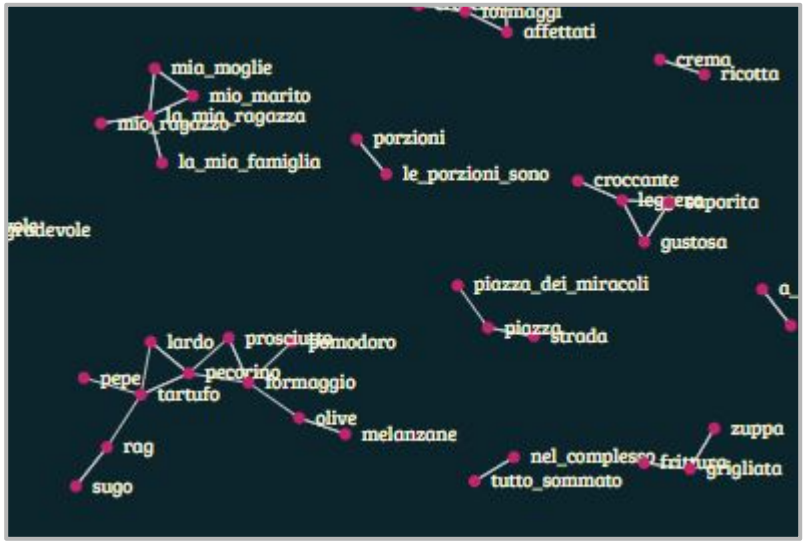
The impact of training data

The source on which a model is trained determines what semantic is captured.

WIKI	BOOKS	WIKI	BOOKS
sega		chianti	
dreamcast	motosega	radda	merlot
genesis	seghe	gaiole	lambrusco
megadrive	seghetto	montespertoli	grignolino
snes	trapano	carmignano	sangiovese
nintendo	smerigliatrice	greve	vermentino
sonic	segare	castellina	sauvignon

Exploring embeddings





Word embeddings to documents

How to represent a document using word embeddings?

- average
- max
- max+min (double length)
- Doc2Vec
- As a layer in a more complex neural network

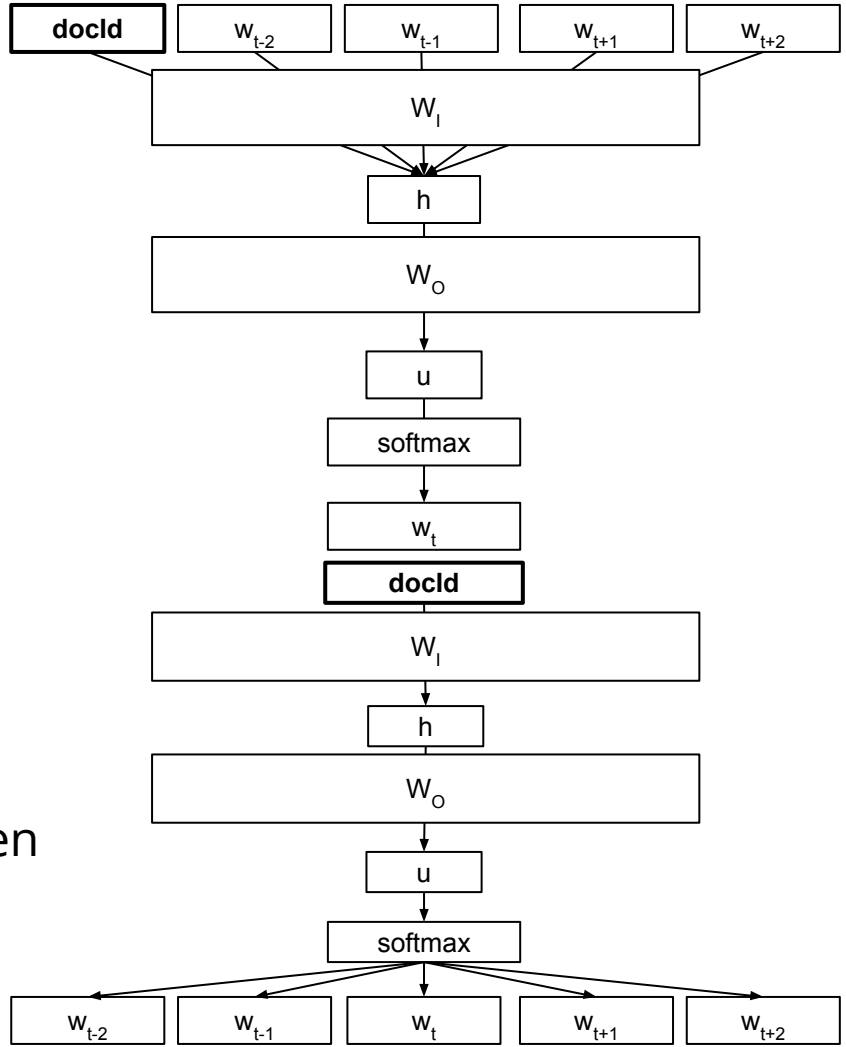
Doc2Vec

Proposed by Le and Mikolov, Doc2Vec extends Word2Vec by adding input dimensions for identifiers of documents.

W_I matrix is $(|D|+|F|)\cdot|h|$.

Documents ids are projected in the same space of words.

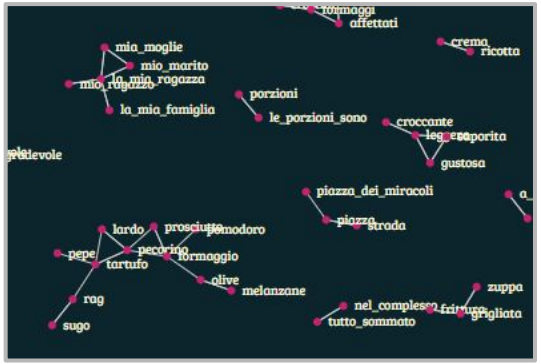
The trained model can be used to infer document embeddings for previously unseen documents - by passing the words composing them.



Exploring embeddings

Documents embedding can be used as vectorial representations of documents in any task.

When the document id is associated to more than one actual document (e.g., id of a product with multiple reviews), Doc2Vec is a great tool to model similarity between objects with multiple descriptions.



Embeddings in neural networks

An embedding layer in neural networks is typically the first layer of the network.

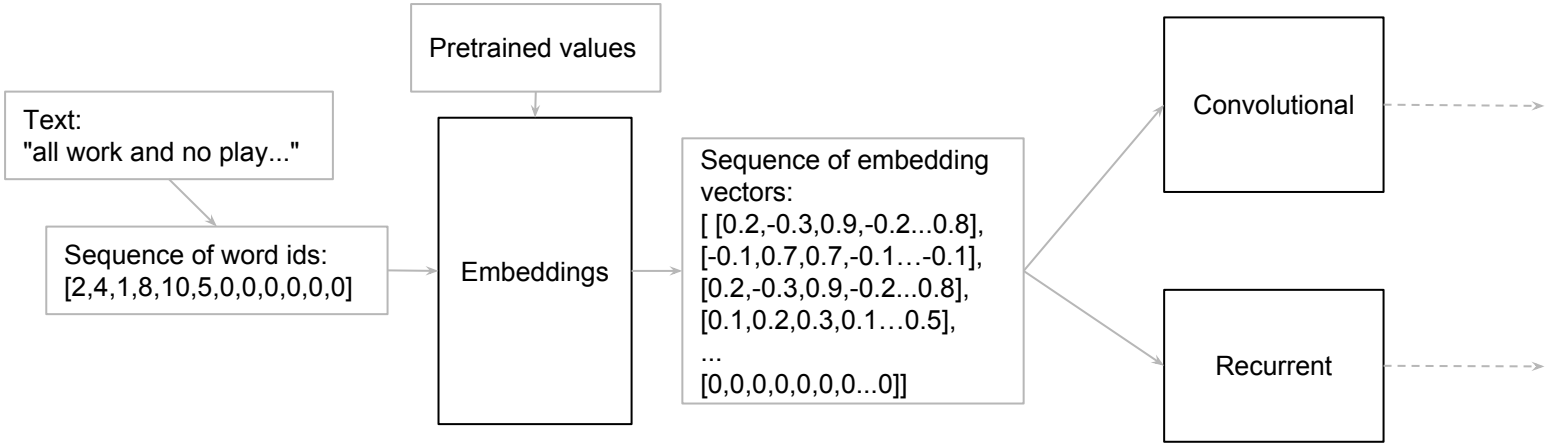
It consists of a matrix W of size $|F| \cdot n$, where n is the size of the embedding space.

It maps words to dense representations.

It can be initialized with random weights or pretrained embeddings.

During learning weights can be kept fixed (it makes sense only when using pre-trained weights) or updated, to adapt embeddings to the task.

Embeddings in neural networks



Example:
https://github.com/fchollet/keras/blob/master/examples/imdb_cnn.py

Another example:
https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/