Language Models

Andrea Esuli

Statistical Language Model

A statistical language model is a *probability distribution* P over sequences of terms.

Given a document d that is composed of a sequence of words $w_1w_2w_3$, we can define:

$$P(d) = P(w_1w_2w_3) = P(w_1)P(w_2|w_1)P(w_3|w_1w_2)$$

Depending on the assumptions we make on the probability distribution, we can create statistical model of different complexity.

The formula above makes no assumptions and can exactly model any language, yet it is impractical because it requires to learn the probability of any sequence in the language.

Unigram model

A unigram model assumes a *statistical independence* between words, i.e., the probability of d is the product of the probabilities of its words:

$$P(d) = P(w_1 w_2 w_3) = P(w_1)P(w_2|w_1)P(w_3|w_1 w_2)$$

$$= P(w_1)P(w_2)P(w_3) = \Pi_i P(w_i)$$

The bayesian classifier that uses this model is called *naïve* for this reason. Usually the models use the logs of the probabilities to work in a linear space:

$$log(\Pi_i P(w_i)) = \Sigma_i \, log(P(w_i))$$

Smoothing, e.g., add one to all frequencies, is used to avoid zero probabilities.

Bigram model

A bigram model assumes a *statistical dependence* of a word from the preceding one:

$$P(d) = P(w_1 w_2 w_3) = P(w_1)P(w_2|w_1)P(w_3|w_1 w_2)$$

$$= P(w_1)P(w_2|w_1)P(w_3|w_2) = \Pi_i P(w_i|w_{i-1})$$

This simple addition is already able to capture a good amount of language regularities.

In general, the longer the n-gram we adopt for the model:

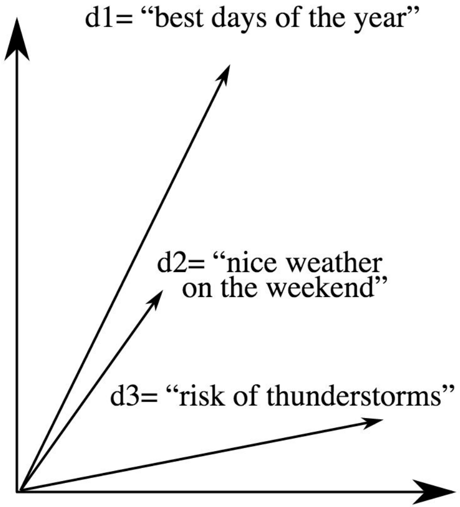
- the more semantic is captured;
- the less statistical significant is the model (memorization/generalization).

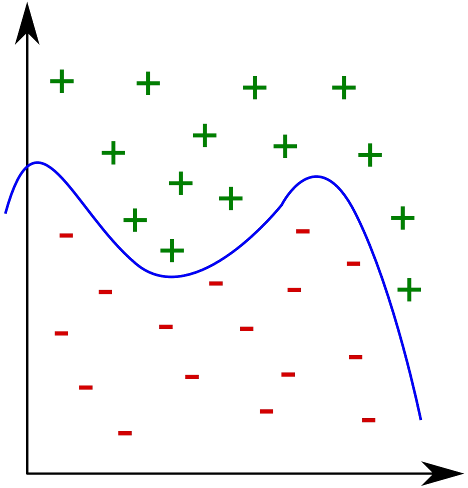
Vector Space Model

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

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


d1= “best days of the year”
d2= “nice weather on the weekend”
d3= “risk of thunderstorms”



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:

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

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:

$$d = \texttt{'you played a good game'}$$

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

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'

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

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.*

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.

Context$_{-2,+2}$('cake') = {['cook','the','twenty', 'minutes']}
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.

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* (i.e. word embeddings).

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

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

We force features to share dimensions on a reduced dense space

↓

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

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 $n{\cdot}n$, with values $\sigma_1, \sigma_2 \ldots \sigma_n$

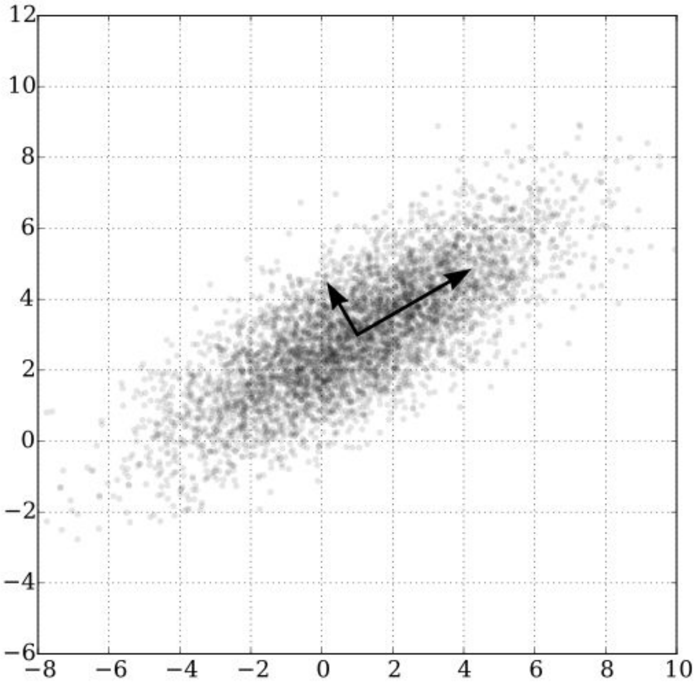
V is an orthonormal matrix of size $n{\cdot}n$, V^* is its conjugate transpose

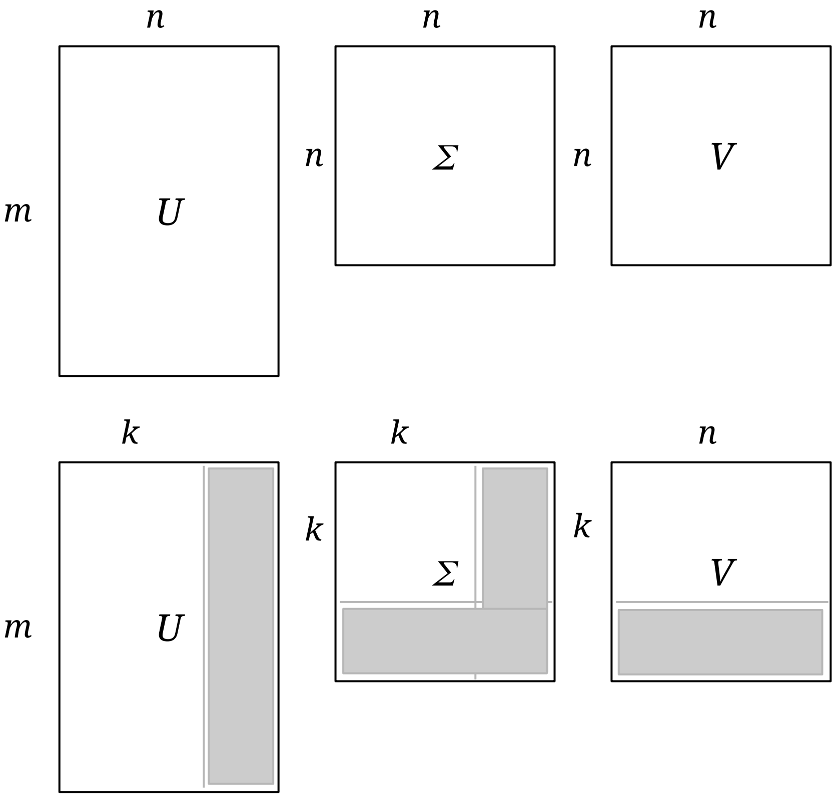
$\sigma_1, \sigma_2 \ldots \sigma_n$ of Σ are the singular values of X, sorted by decreasing magnitude.

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

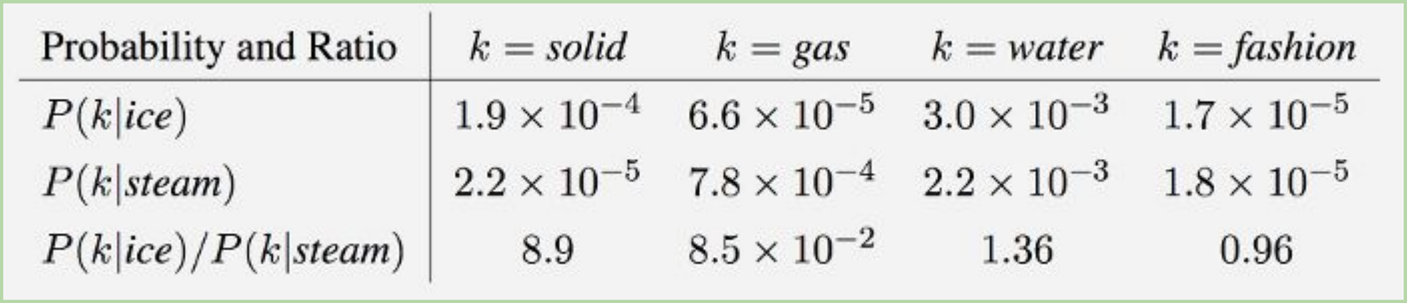




By Nicoguaro (Own work) [CC BY 4.0], via Wikimedia Commons

GloVe

GloVe: Global Vectors for Word Representation is a count-based model that implicitly 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

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

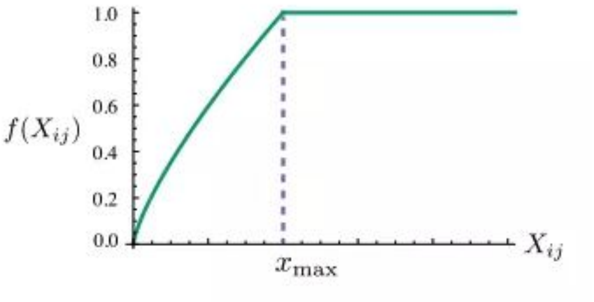
← eqn 8 of GloVe paper

GloVe

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

← eqn 8 of GloVe paper

Weighting function to filter out rare co-occurrences and to avoid frequent ones to dominate



embedding vectors (compressed)

co-occurrence matrix (sparse)

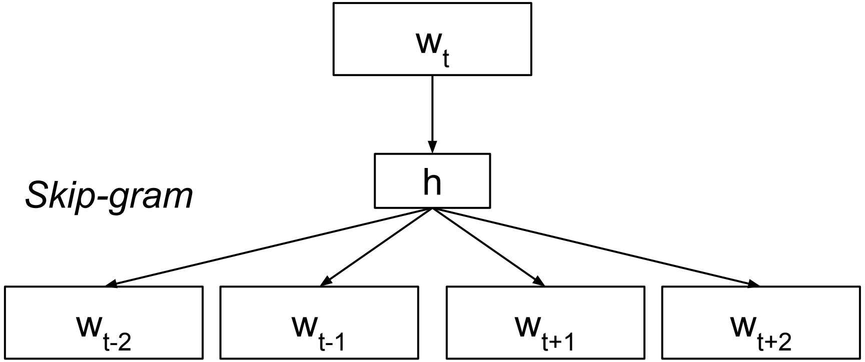
This part implements, as a least square problem, the equation that defines the model:

$$F(w_i, w_j, \tilde{w_k}) \approx \frac{P_{ij}}{P_{jk}}$$

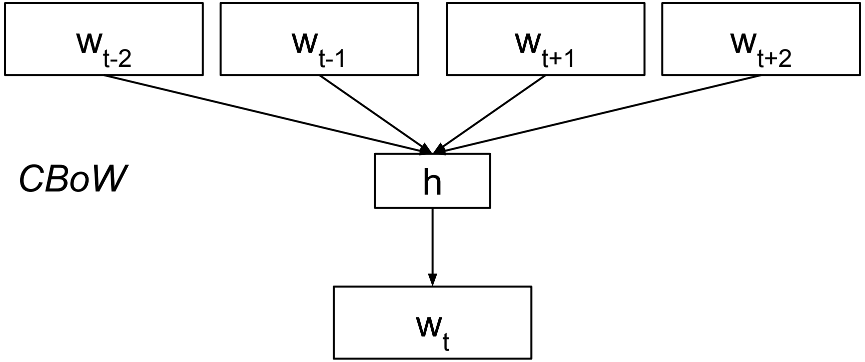
Word2Vec

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.



Skip-gram



CBoW

Word2Vec

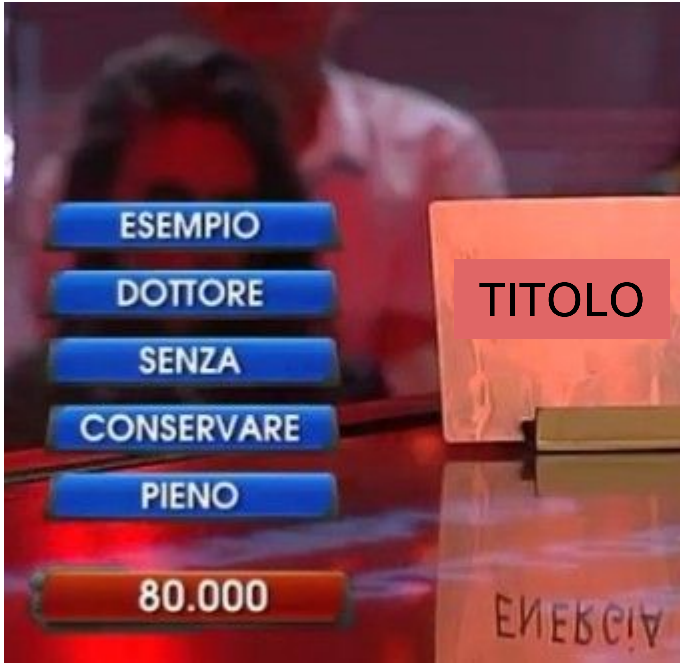
Embeddings are a byproduct of the word prediction task.

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

The context window size ranges between two and five words before and after the central word.

Longer windows capture more semantic, less syntax.

A typical size for h is 200~300.



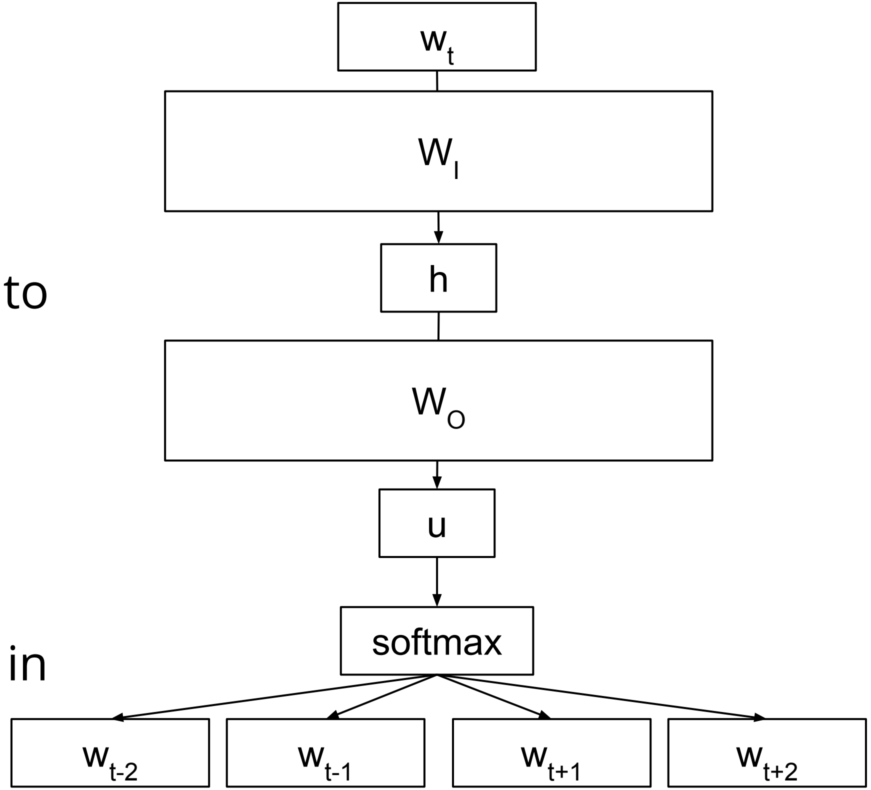
Skip-gram

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.



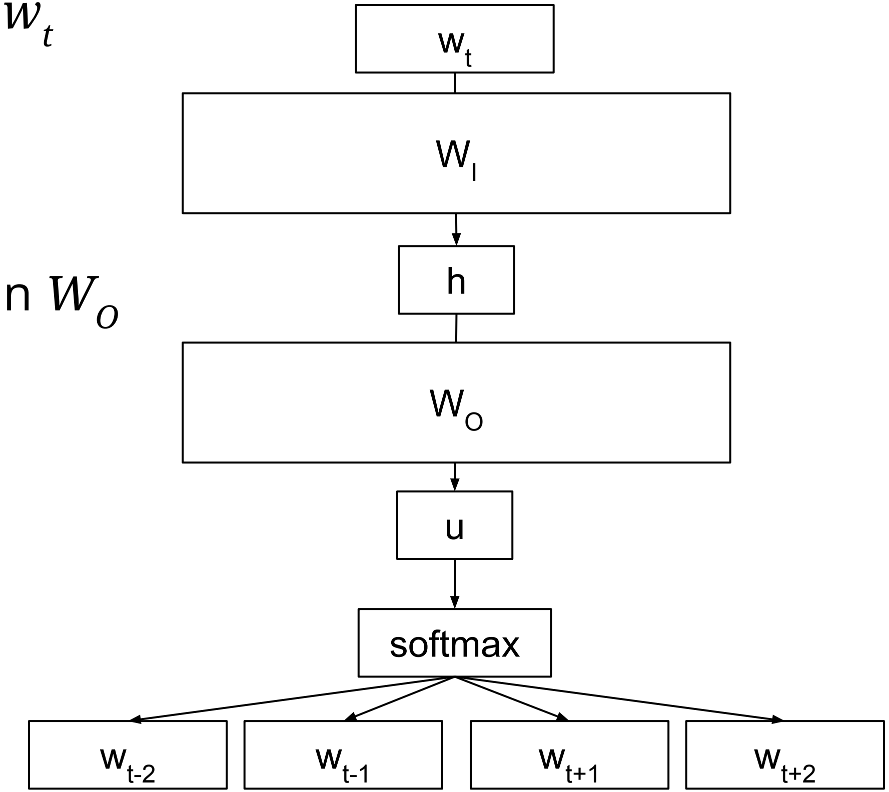
Skip-gram

$h = w_t W_I \leftarrow$ 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 probability distribution y:

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



Skip-gram

Loss: $- \mathit{log}\ p(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} | w_t) =$

$= - \mathit{log}\ \Pi_{c \in C}\, \mathit{exp}(u_c) / \sum_{j \in F} \mathit{exp}(u_j) =$

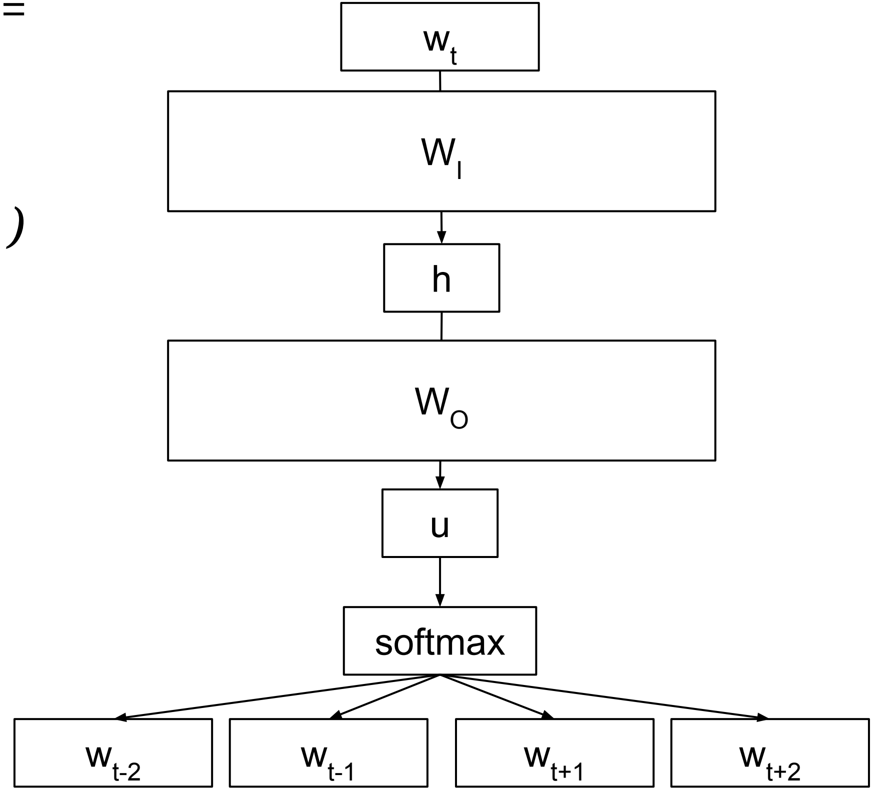
$= - \sum_{c \in C} \mathit{exp}(u_c) + C\ \mathit{log} \sum_{j \in F} \mathit{exp}(u_j)$

i.e., maximize probability of context

$- \sum_{c \in C} \mathit{exp}(u_c)$

and minimize probability of the rest

$+ C\ \mathit{log} \sum_{j \in F} \mathit{exp}(u_j)$



Negative sampling

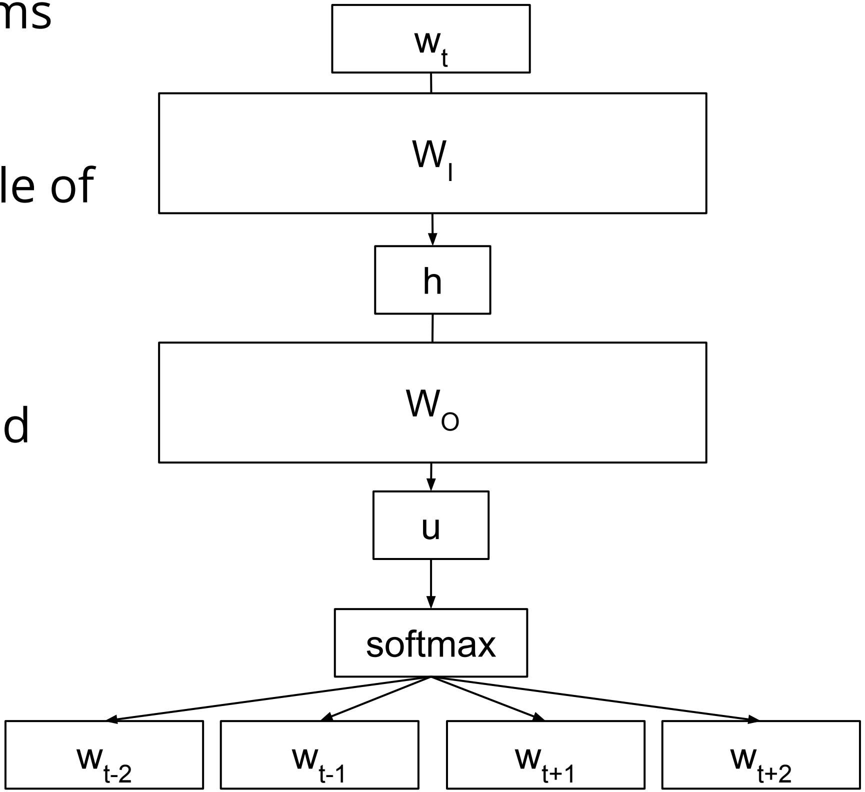
The $log \sum_{j \in F} exp(u_j)$ factor has a lots of terms and it is costly to compute.

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:

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



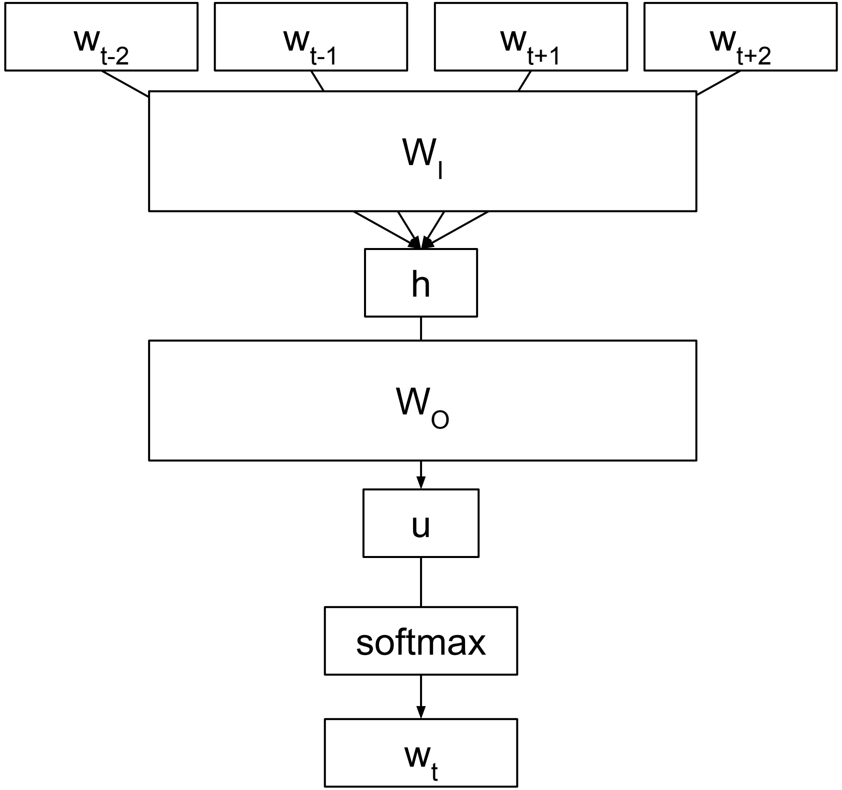
CBoW

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 word embedding w_t in W_O



Which model to choose?

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)
```

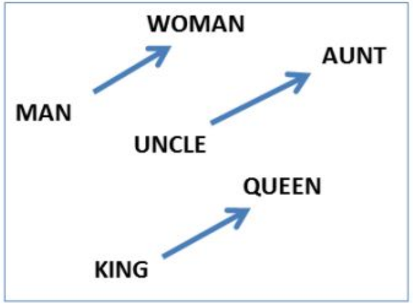
This is a clean implementation of skip-grams using pytorch.

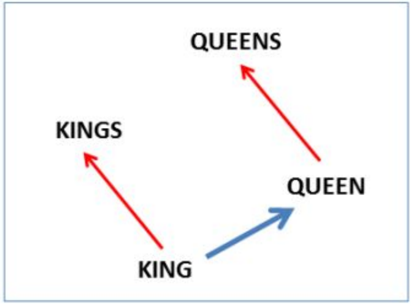
Which embeddings?

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	?

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

$$a : b = c : d$$

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

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

FastText word representation

FastText extends the W2V embedding model to *ngrams* of the words.

The word "goodbye" is also represented with a set of ngrams:

"<go" (star of word), "goo","ood", "odb", "dby", "bye", "ye>" (end of word)

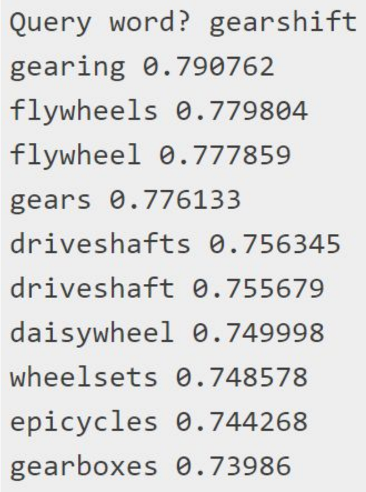
The length of the ngram is a parameter.

Typically all ngrams of length from 3 to 6 are included.

FastText word representation

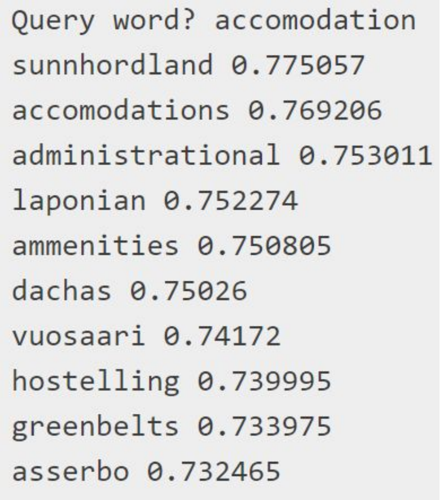
The embedding of a word is determined as the sum of the embedding of the word and of the embedding of its ngrams.

Subword information allows to give an embedding to OOV words.

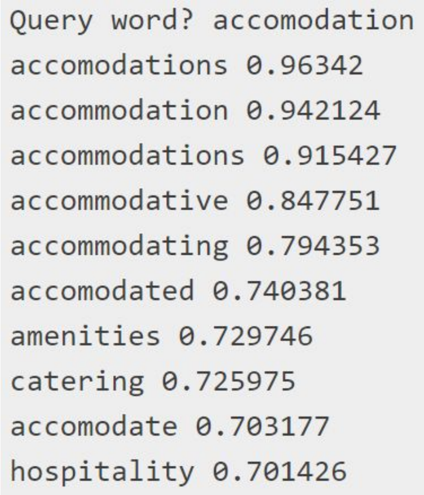


```
Query word? gearshift
gearing 0.790762
flywheels 0.779804
flywheel 0.777859
gears 0.776133
driveshafts 0.756345
driveshaft 0.755679
daisywheel 0.749998
wheelsets 0.748578
epicycles 0.744268
gearboxes 0.73986
```

Subword information improves the quality of misspelled words.



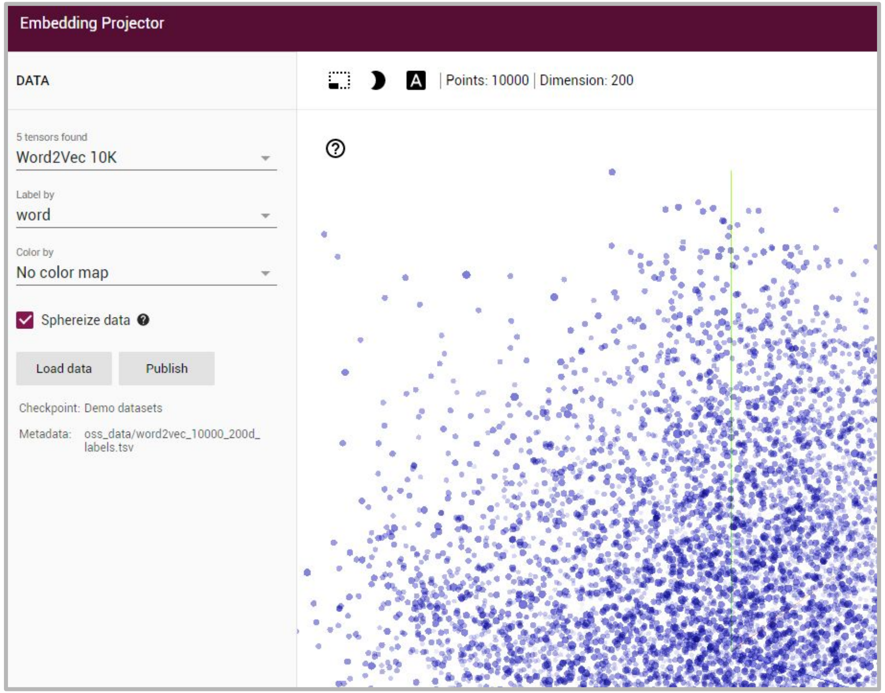
```
Query word? accomodation
sunnhordland 0.775057
accomodations 0.769206
administrational 0.753011
laponian 0.752274
ammenities 0.750805
dachas 0.75026
vuosaari 0.74172
hostelling 0.739995
greenbelts 0.733975
asserbo 0.732465
```



```
Query word? accomodation
accomodations 0.96342
accommodation 0.942124
accommodations 0.915427
accommodative 0.847751
accommodating 0.794353
accomodated 0.740381
amenities 0.729746
catering 0.725975
accomodate 0.703177
hospitality 0.701426
```

Pretrained embeddings for 200+ languages.

Exploring embeddings



Embedding Projector

DATA

5 tensors found
Word2Vec 10K

Label by
word

Color by
No color map

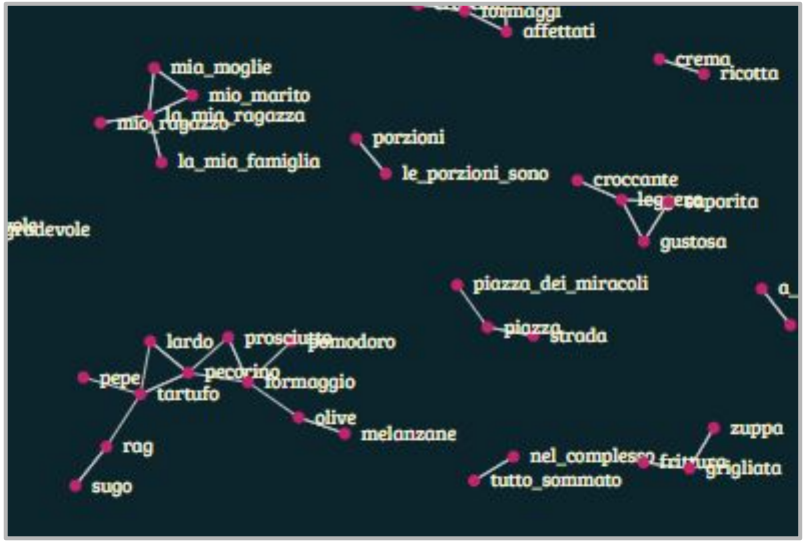
☑ Sphereize data

Load data | Publish

Checkpoint: Demo datasets

Metadata: oss_data/word2vec_10000_200d_labels.tsv

| Points: 10000 | Dimension: 200



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

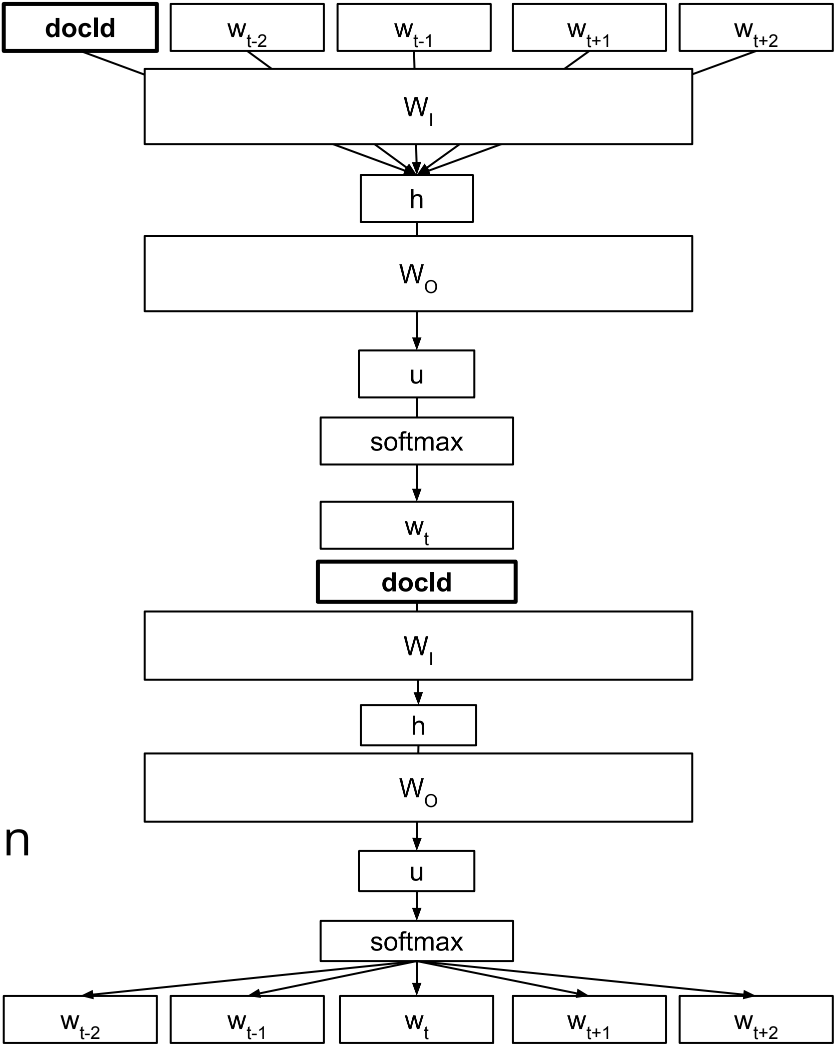
Doc2Vec

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.

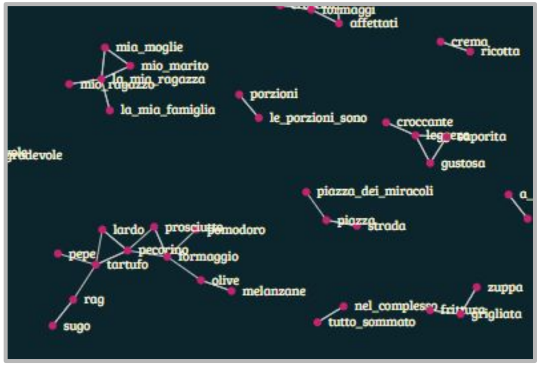
The trained model can be used to infer document embeddings for previously unseen documents - by passing the words composing them.



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 pretrained weights) or updated, to adapt embeddings to the task.

OOV words and padding

LMs that use a vocabulary do not model out-of-vocabulary words.

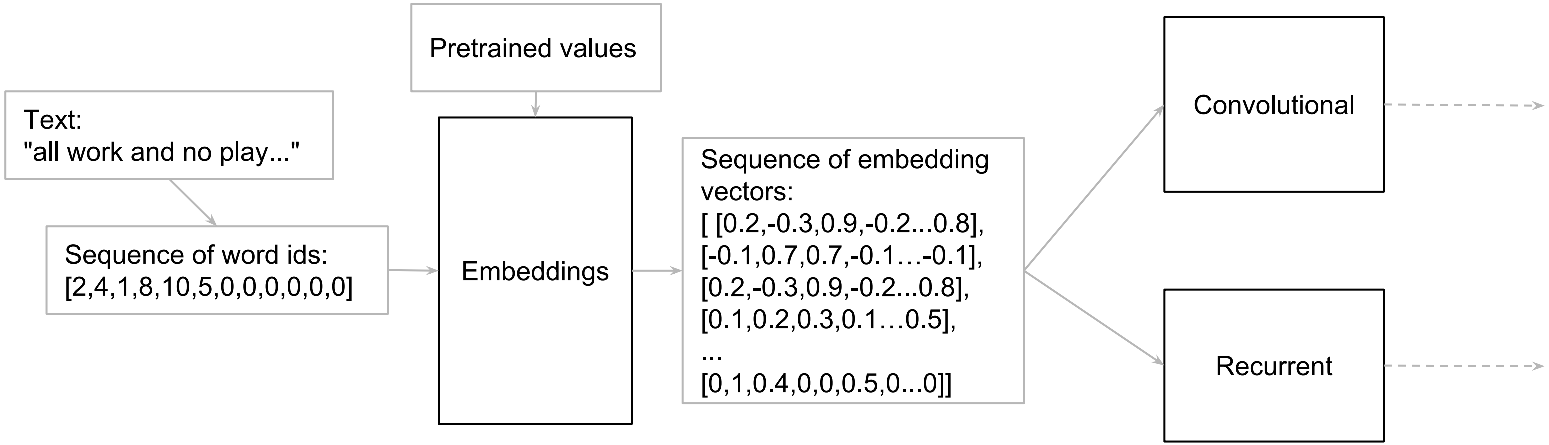
Add a special *unknown* word (and embedding) for such words, to be learned during training.

NNs usually process examples in *batches*, i.e., set of k examples.

Input sentences in a batch are usually required to be of the same length.

For this reason a special *padding* word (and embedding) is added before/after (be consistent!) words of shorter sentence to match the length of the longest one.

Embeddings in neural networks



Pretrained values

Text:
"all work and no play..."

Sequence of word ids:
[2,4,1,8,10,5,0,0,0,0,0,0]

Embeddings

Sequence of embedding vectors:
[[0.2,-0.3,0.9,-0.2...0.8],
[-0.1,0.7,0.7,-0.1…-0.1],
[0.2,-0.3,0.9,-0.2...0.8],
[0.1,0.2,0.3,0.1…0.5],
...
[0,1,0.4,0,0,0.5,0...0]]

Convolutional

Recurrent

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

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

Recurrent Neural Networks

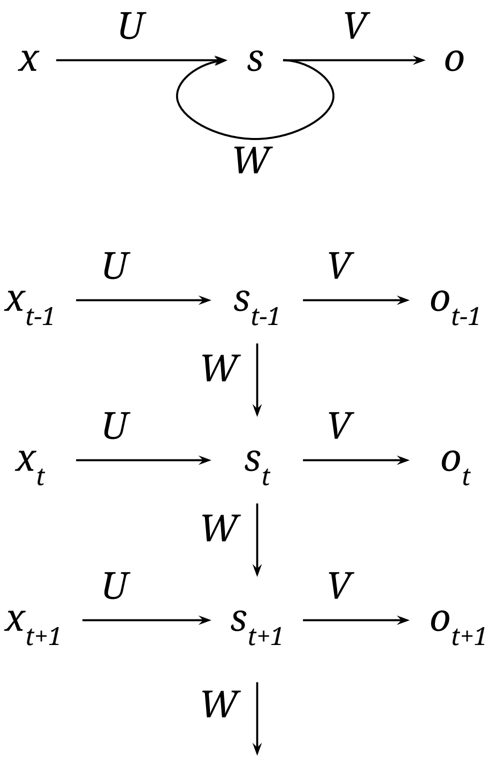
A Recurrent Neural Network (RNN) is a neural network in which connections between units form a directed cycle.

Cycles allow the network to have a memory of previous inputs, combining it with current input.

RNNs are fit to process sequences, such as text.

Text can be seen as a sequence of values at many different levels: characters, words, phrases...

Suggested read

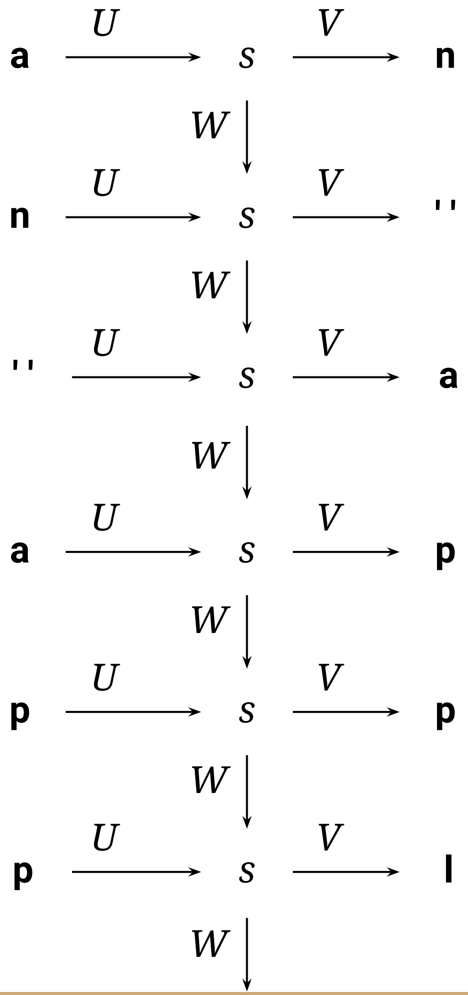


Char-level LM & text generation

RNNs are key tools in the implementation of many NLP applications, e.g., machine translation, summarization, or image captioning.

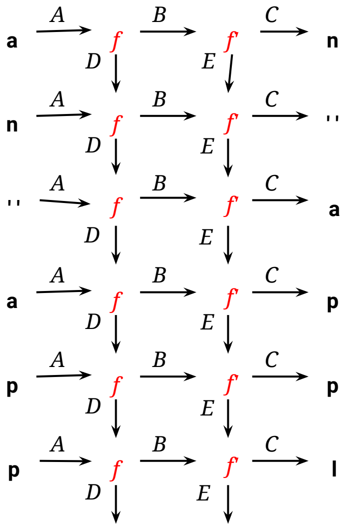
A RNN can be used to learn a language model that predicts the next character from the sequence of previous ones.

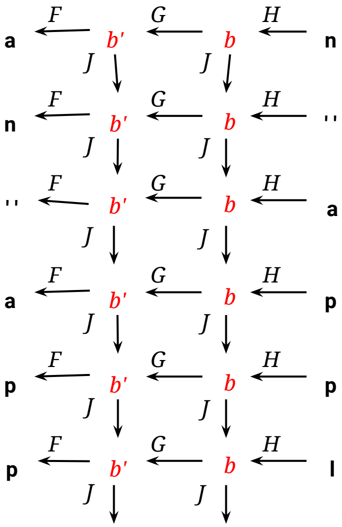
The typical RNN node that is used is an Long Short Term Memory (LSTM), which is robust to typical issues of RNNs.



ELMo

ELMo (Embeddings from Language Models) exploits the *hidden states* of a *deep*, *bi-directional*, *character-level* LSTM to give *contextual* representations to words in a sentence.





The embedding for a word is a *task-specific* weighted sum of the concatenation of the f,b vectors for each level of the LSTM, e.g.:

$$v(token) = w\,[f,\,b]_{token} + w'\,[f'\,,b']_{token}$$

CoVe, GPT, BERT

Other recent proposals exploit RNN and/or other mechanisms such as Attention, to assign contextualized embeddings to words:

- CoVE (Context Vector), uses the hidden state of a Machine Translation network.
- GPT (Generative Pre-Training), learns a language model using attention and then fine tunes it to the task of interest.
- BERT (Bidirectional Encoder Representations from Transformers), extends GPT with bidirectional language modeling.