DD2418 Language Engineering
6a: Lexical semantics

Johan Boye, KTH

Lexical semantics

What is the *meaning* of a word?

- ... i.e. what kind of object/entity is the meaning of a word?

Can we represent the meaning of a word in a computer program?

- ... and if so, how?

What is the meaning of a word?

Linguistics answer:

- Words (*signifiers*) denote *concepts* (the *signified*)
  - e.g. “dog” denotes the concept of a dog
- A concept has an extension and an intention
  - e.g. extension = all dogs in the world
  - intention = the abstract idea of a dog

Words and senses

A word can have several meanings, or *senses*. This can be due to one of a number of linguistic processes:

Homonymy. unrelated senses

- *bear* (the animal), and the verb *to bear*.

Polysemy: related senses

- *move* to the next room, *move furniture*
- a *good* man, a *good* painter

Metaphors

- *He got the idea.*
- *He bought the argument.*

Semantic relations

Words (or rather word senses) can have various semantic relationships to one another.

- synonyms, e.g. *good-nice*
- antonyms, e.g. *good-bad*
- hypernyms and hyponyms (is-a relations), e.g. *animal-dog*
- meronyms (part-of relations), e.g. *tree-branch*
- metonyms, e.g. *"Agatha Christie is very exciting"*

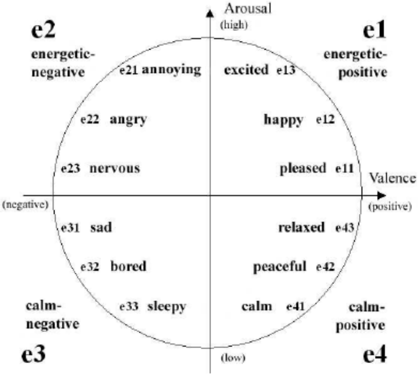
Semantic fields

Words that are not synonyms, but still related because they are used in the same domain.

For instance:
school, teacher, pupil, classroom, course, schedule, etc.

Sentiment

Words (not only adjectives) often comes with different connotations, which can be individually and culturally dependent.



Lexical semantics

We are looking for a way to represent meaning of words in a way usable for a computer program.

One possibility is to use an on-line *taxonomy* like *WordNet*, developed at Princeton.

WordNet specifies *hyponym* (is-a) relations, and *synonym sets*.

It has a web interface, and is also integrated into NLTK.

WordNet

Noun

- S: (n) **simple** (any herbaceous plant having medicinal properties)
- S: (n) simpleton, **simple** (a person lacking intelligence or common sense)

Adjective

- S: (adj) **simple** (having few parts; not complex or complicated or involved) *"a simple problem"; "simple mechanisms"; "a simple design"; "a simple substance"*
- S: (adj) elementary, **simple**, uncomplicated, unproblematic (easy and not involved or complicated) *"an elementary problem in statistics"; "elementary, my dear Watson"; "a simple game"; "found an uncomplicated solution to the problem"*
- S: (adj) bare, mere, **simple** (apart from anything else; without additions or modifications) *"only the bare facts"; "shocked by the mere idea"; "the simple passage of time was enough"; "the simple truth"*
- S: (adj) childlike, wide-eyed, round-eyed, dewy-eyed, **simple** (exhibiting childlike simplicity and credulity) *"childlike trust"; "dewy-eyed innocence"; "listened in round-eyed wonder"*
- S: (adj) dim-witted, **simple**, simple-minded (lacking mental capacity and subtlety)
- S: (adj) **simple**, unsubdivided ((botany) of leaf shapes; of leaves having no divisions or subdivisions)
- S: (adj) **simple** (unornamented) *"a simple country schoolhouse"; "her black dress--simple to austerity"*

WordNet

Verb

- S: (v) **bare** (lay bare) *"bare your breasts"; "bare your feelings"*
- S: (v) publicize, publicise, air, **bare** (make public) *"She aired her opinions on welfare"*
- S: (v) denude, **bare**, denudate, strip (lay bare) *"denude a forest"*

Adjective

- S: (adj) **bare**, au naturel, naked, nude (completely unclothed) *"bare bodies"; "naked from the waist up"; "a nude model"*
- S: (adj) **bare**, scanty, spare (lacking in magnitude or quantity) *"a bare livelihood"; "a scanty harvest"; "a spare diet"*
- S: (adj) unsheathed, **bare** (not having a protective covering) *"unsheathed cables"; "a bare blade"*
- S: (adj) **bare** (lacking its natural or customary covering) *"a bare hill"; "bare feet"*
- S: (adj) **bare**, marginal (just barely adequate or within a lower limit) *"a bare majority"; "a marginal victory"*
- S: (adj) **bare**, mere, simple (apart from anything else; without additions or modifications) *"only the bare facts"; "shocked by the mere idea"; "the simple passage of time was enough"; "the simple truth"*
- S: (adj) **bare**, unfinished (lacking a surface finish such as paint) *"bare wood"; "unfinished furniture"*
- S: (adj) **bare**, barren, bleak, desolate, stark (providing no shelter or sustenance) *"bare rocky hills"; "barren lands"; "the bleak treeless regions of the high Andes"; "the desolate surface of the moon"; "a stark landscape"*
- S: (adj) **bare**, stripped (having everything extraneous removed including contents) *"the bare walls"; "the cupboard was bare"*
- S: (adj) plain, **bare**, spare, unembellished, unornamented (lacking embellishment or ornamentation) *"a plain hair style"; "unembellished white walls"; "functional architecture featuring stark unornamented concrete"*

Swedish WordNet

Swedish Wordnet

[ ] Sök..

Show Hyponym Tree | Show Hyperonym Tree

Synset

hänga-1 [v] *få att hållas över marken så att föremålet inte rör underlaget; "Fågeln hänger i gardinen"*

Relations

has_hyperonym hänga upp-1 [v]
has_hyponym svinga-1 [v]
has_hyponym dingla-1 [v]
has_hyponym säcka-1 [v]

EQ-Link

Problems with WordNet

Has to be built and maintained by someone.

Subjective, ad-hoc, no (explicit) empirical basis.

Missing many nuances.

Binary distinction: Either a synonym, or not.

Hard to compute word similarity.

DD2418 Language Engineering
6b: Word embeddings

Johan Boye, KTH

Word embeddings

How can we represent the meaning of words?

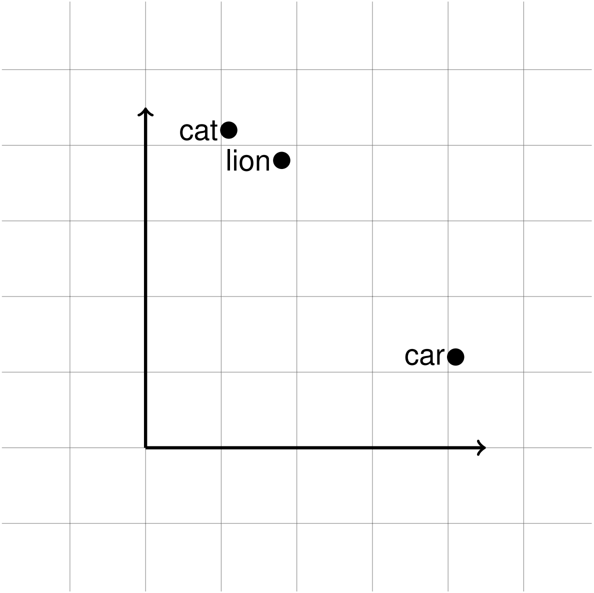
We want words of similar meaning have similar representations.

Idea: View words as points or vectors in a high-dimensional vector space (a "distributed" representation), e.g.:

$$(.003, .25, 1.2, -3.4, \ldots, -.775)$$

Such vectors are often called word embeddings (because they are "embedded" in a vector space).

Language as a vector space



Distributional hypothesis

The distributional hypothesis says that if two words w_1 and w_2 tend to **co-occur with the same words**, then w_1 and w_2 **have a similar meaning.**

Wittgenstein: *The meaning of a word is its use in the language*.

Firth: *You shall know a word by the company it keeps.*

Word-document matrix

One way of creating word vectors is through a *word-document matrix*.

		Documents						
		1	2	3	4	5	...	n
	aalborg	1						
	aback		1		1			
	abandon		1					
Words	...							
	zombie					1		
	zone							
	zurich			1				

Word-document matrix

The word vector for *aback* is highlighted below.

		Documents						
		1	2	3	4	5	...	n
	aalborg	1						
	aback	**0**	**1**	**0**	**1**	**0**	...	**0**
	abandon		1					
Words	...			...				
	zombie					1		
	zone							
	zurich			1				

Word windows

Another possibility would be to look in the immediate context of a word:

. . . I would like $\underbrace{\textbf{to}}_{\text{focus word}}$ know the plans for . . .

Word-word matrix

		Context words			
		know	the	plans	...
	know	0	98	2	...
Focus words	the	98	12	11	...
	plans	2	11	0	...
	...				$\ddots$

Meaning captured by word vectors

What 'meaning' do such word vectors capture?

Vectors obtained from word-document matrices:

- words with similar vectors tend to belong to the same semantic field
- e.g. *hospital* and *nurse*

Vectors obtained from word-word matrices:

- words with similar vectors tend to have the same part-of-speech
- both syntactic and semantic relationships captured

The problem with such word vectors is that they are very high-dimensional and sparse.

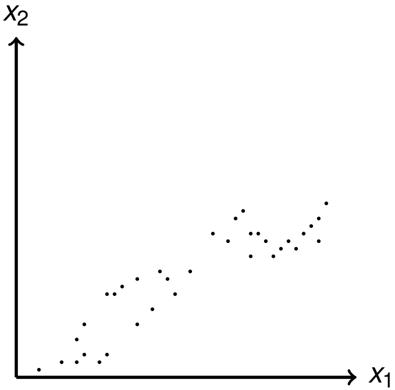
Dimensionality reduction

We can reduce the dimensionality of a matrix while keeping as much of the variance as possible, using

- t-SNE (t-distibuted stochastic neighbor embedding), or
- SVD (singular value decomposition), a particular kind of matrix factorization

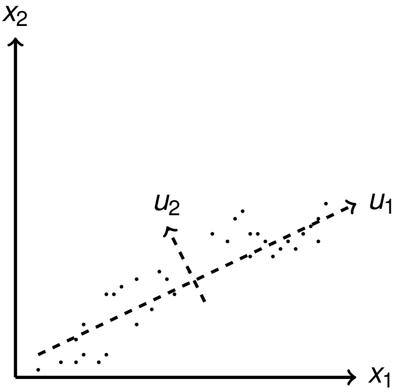
SVD-factorization of the word-document matrix leads to so-called Latent Semantic Analysis (LSA).

Dimensionality reduction



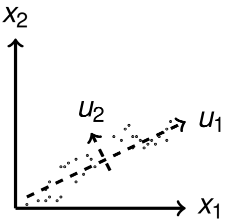
Keep the dimension which has the highest variance.

Dimensionality reduction



Even better: Change bases into u_1 and u_2. Keep the dimension with the highest variance.

Dimensionality reduction



On the other hand: u_1 and u_2 do not correspond to contexts like x_1 and x_2, but rather abstract concepts whose exact nature is unknown.

Visualizations from a bigger corpus

Figure 10: Multidimensional scaling of three verb semantic classes.

Rohde et al. (2005) An improved model of semantic similarity based on lexical co-occurrence. *Communications of the ACM*, 8(627-633), 116.

DD2418 Language Engineering
6c: Random indexing

Johan Boye, KTH

Problems with Latent Semantic Analysis

Computationally costly to factor the large matrix (which might be $10^6 \times 10^6$).

The method is not incremental. Every time you get more data, you again need to factor a large matrix.

Random indexing

Random indexing is a simpler and more flexible approach to contructing word vectors.

Assign 2 vectors to every word w:

- a random vector (or index vector) $r(w)$ of dimensionality d (typically $d = 2000$).
- a context vector $c(w)$ of dimensionality d as well.

$r(w)$ is initially filled with n non-zero values

- n is typically 100
- a non-zero value is either 1 or -1, randomly selected

$c(w)$ is initially filled with 0.

Random indexing

Use the sliding context window idea again:

$$\ldots \text{ I } \text{would like } \underbrace{\textbf{to}}_{\text{focus word}} \text{ know the } \text{plans for } \ldots$$

In each step, compute:

$$c(w_i) += (r(w_{i-2}) + r(w_{i-1}) + r(w_{i+1}) + r(w_{i+2}))$$

Do this for a lot of text. In the end, a small distance between $c(w_i)$ and $c(w_k)$ indicates that w_i and w_k are semantically similar.

Random indexing, example

Word	would	like	to	know	the
Random	(1 0 -1 0 0 1 0)	(0 1 0 0 -1 -1 1)	(0 0 0 1 0 1 1)	(0 -1 0 1 1 0 0)	(1 0 1 -1 0 0 -1)
Context	(0 -1 -5 -2 1 -3)	(-5 3 2 1 -3 -5 -5)	(-4 2 0 -3 1 2 2)	(-3 5 3 -1 -4 1 4)	(0 4 0 1 1 -1 0)

Compute one step of random indexing, using the vectors above and the situation below:

. . . I would like $\underbrace{\textbf{to}}_{\text{focus word}}$ know the plans for . . .

Random indexing, example

Word	would	like	to	know	the
Random	(1 1 -1 0 0 1 0)	(0 1 0 0 -1 -1 1)	(0 0 0 1 0 1 1)	(0 -1 0 1 1 0 0)	(1 0 1 -1 0 0 -1)
Diff			(2 1 0 0 0 0 0)		
Context	(0 -1 -5 -2 1 -3)	(-5 3 2 1 -3 -5 -5)	(-4 2 0 -3 1 2 2)	(-3 5 3 -1 -4 1 4)	(0 4 0 1 1 -1 0)

Compute one step of random indexing, using the vectors above and the situation below:

. . . I would like **to** know the plans for . . .

focus word

Word	would	like	to	know	the
Random	(1 1 -1 0 0 1 0)	(0 1 -1 0 -1 -1 1)	(0 0 0 1 0 1 1)	(0 -1 0 -1 1 1 0)	(1 0 1 -1 0 1 -1)
Context	(0 -1 -5 -2 1 -3)	(-5 3 2 1 -3 -5 -5)	(-2 3 0 -3 1 2 2)	(-3 5 3 -1 -4 1 4)	(0 4 0 1 1 -1 0)

Random indexing

Random indexing was devised in 2000 by Pentti Kanerva.

Empirically, it has shown to work very well. But there is no (known) theoretical justification for it.

DD2418 Language Engineering
6d: word2vec

Johan Boye, KTH

word2vec

word2vec is a software package, released in 2013 by Tomas Mikolov.

word2vec could build word vectors for any language:

- quick, efficient
- word vectors trained with Word2Vec have many amazing features:

$$v(France) - v(Paris) + v(Rome) = v(Italy)$$
$$v(king) - v(man) + v(woman) = v(queen)$$

Starting point of the neural network "revolution" in language engineering.

word2vec

Imagine (again) a context window sliding over text:

$$\ldots \text{ I } \boxed{\text{would like}} \; \underbrace{\boxed{\textbf{to}}}_{\substack{\text{focus}\\ \text{word}}} \; \boxed{\text{know the}} \text{ plans for } \ldots$$

Then: Train a classifier that either

- can predict the focus word from the context words (continuous bag-of-words model),
- or vice versa (skipgram model)

(Actually, we don't care about this classifier. But the trainable parameters of the model will form our word vectors.)

word2vec

Each word has two vectors:

- w (the focus word vector),
- $\widetilde{w}$ (the context word vector).

The dimensionality of the vectors is a hyperparameter of the method (typically 50-300).

The trainable parameters of the prediction model we are going to train are the vectors themselves.

The Skipgram prediction task

Skipgram = predict the context words from the focus word.

$$\ldots \text{ I } \boxed{\text{would like}} \ \underbrace{\boxed{\textbf{to}}}_{\text{focus word}} \ \boxed{\text{know the}} \text{ plans for } \ldots$$

We write $P(u|v)$ to mean "the probability that u appears in the context of v".

For instance, we want $P(\textbf{would}|\textbf{to})$ to be close to 1.

We set $P(u|v) = \sigma(\widetilde{u}^T v)$, where σ is the logistic function.

That is, we want to learn the vectors $\widetilde{\textbf{would}}$ and **to**, so that $(\widetilde{\textbf{would}}^T \cdot \textbf{to})$ is as big as possible.

Negative sampling

. . . I would like $\underbrace{\textbf{to}}_{\text{focus word}}$ know the plans for . . .

We also need negative samples, i.e. examples of words not appearing in the context of the focus word (k negative samples for every positive).

For instance, we can select "text" as a negative sample for "to".

Then we want $P(\textbf{text}|\textbf{to}) = \sigma(\widetilde{\textbf{text}}^T \cdot \textbf{to})$ to be close to 0.

That is, $1 - P(\textbf{text}|\textbf{to}) = \sigma(-\widetilde{\textbf{text}}^T \cdot \textbf{to})$ should be close to 1.

Choosing negative samples

Mikolov et al: Randomly choose negative samples according to the unigram distribution P_u, raised to 0.75.

$$P_s(w) = \frac{P_u(w)^{0.75}}{\sum_{w'} P_u(w')^{0.75}}$$

The "0.75" upsamples rare words a bit, e.g.:

$$0.01^{0.75} = 0.03$$
$$0.99^{0.75} = 0.992$$

Training procedure (1)

First:

- obtain a lot of running text
- decide how long vectors we want (e.g. 50)
- for each word in the vocabulary create one focus vector and one context vector, randomly initialized
- decide on the size of the context window (e.g. 2)

Training procedure (2)

We will now obtain positive and negative samples by moving through the text.

$$\ldots \text{ I } \boxed{\text{would like}} \; \underbrace{\boxed{\textbf{to}}}_{\substack{\text{focus}\\ \text{word}}} \; \boxed{\text{know the}} \text{ plans for } \ldots$$

Positive examples: (to,would), (to,like), (to,know), (to,the)

For each positive example, sample 5-20 negative examples.

Then move word window one step, and collect more positive and negative examples.

Semi-supervised learning: You get the labels of data points for free!

Loss function

We want to maximize the probability of the positive examples pos, and minimize the probability of the negative examples neg.

Equivalently, we want to minimize the loss defined as:

$$-\sum_{(v,\widetilde{u})\in \text{pos}} \log \sigma(\widetilde{u}^T v) - \sum_{(v,\widetilde{u})\in \text{neg}} \log \sigma(-\widetilde{u}^T v) =$$

$$\sum_{(v,\widetilde{u})\in \text{pos}} \log(1 + e^{-\widetilde{u}^T v}) + \sum_{(v,\widetilde{u})\in \text{neg}} \log(1 + e^{\widetilde{u}^T v})$$

Note that both $\widetilde{u}$ and v are trainable parameters.

Gradient of the loss function (1)

Gradient of the loss function w.r.t. a particular v:

$$\frac{\partial}{\partial v}[\sum_{(v,\widetilde{u})\in \text{pos}} \log(1+e^{-\widetilde{u}^T v}) + \sum_{(v,\widetilde{u})\in \text{neg}} \log(1+e^{\widetilde{u}^T v})] =$$

$$-\sum_{(v,\widetilde{u})\in \text{pos}} \frac{\widetilde{u}e^{-\widetilde{u}^T v}}{1+e^{-\widetilde{u}^T v}} + \sum_{(v,\widetilde{u})\in \text{neg}} \frac{\widetilde{u}e^{\widetilde{u}^T v}}{1+e^{\widetilde{u}^T v}} =$$

$$-\sum_{(v,\widetilde{u})\in \text{pos}} \widetilde{u}(1-\sigma(\widetilde{u}^T v)) + \sum_{(v,\widetilde{u})\in \text{neg}} \widetilde{u}(1-\sigma(-\widetilde{u}^T v)) =$$

$$\sum_{(v,\widetilde{u})\in \text{pos}} \widetilde{u}(\sigma(\widetilde{u}^T v)-1) + \sum_{(v,\widetilde{u})\in \text{neg}} \widetilde{u}\sigma(\widetilde{u}^T v)$$

Note that all vectors are parameters, so we also need to compute the gradient w.r.t. a particular $\widetilde{u}$. (see next slide)

Gradient of the loss function (2)

Gradient of the loss function w.r.t. a particular $\widetilde{u}_i$.

If $\widetilde{u}_i$ is in the context of v, i.e. $(v, \widetilde{u}_i)$ is a positive example:

$$\frac{\partial}{\partial \widetilde{u}_i}[\sum_{(v,\widetilde{u})\in \text{pos}} \log(1 + e^{-\widetilde{u}^T v}) + \sum_{(v,\widetilde{u})\in \text{neg}} \log(1 + e^{\widetilde{u}^T v})] =$$

$$-\frac{ve^{-\widetilde{u}_i{}^T v}}{1 + e^{-\widetilde{u}_i{}^T v}} = -v(1 - \sigma(\widetilde{u}_i{}^T v)) = v(\sigma(\widetilde{u}_i{}^T v) - 1)$$

If $\widetilde{u}_i$ is not in the context of v, i.e. $(v, \widetilde{u}_i)$ is a negative example:

(*left as an exercise*)

Training procedure (3)

We obtain positive and negative samples by moving through the text.

$$\ldots \text{ I } \boxed{\text{would like}} \; \underbrace{\boxed{\textbf{to}}}_{\substack{\text{focus}\\\text{word}}} \; \boxed{\text{know the}} \text{ plans for } \ldots$$

Positive examples: (to,would), (to,like), (to,know), (to,the)

Negative examples could be: (to, text), (to, giraffe), **...**

Use gradient descent to update the vectors **to**, **would**, **like**, **know**, . . . (all context vectors above)

Then move word window one step, and collect more positive and negative examples.

word2vec summary

Some nice ideas used in word2vec:

- Word vectors are obtained as a by-product when learning an artificial task (that we don't care about).
- Semi-supervised learning: Obtaining positive and negative examples for the learning task just by stepping through the text.
- Dot product as similarity between word vectors.
- Learning focus vectors and context vectors simultaneously using gradient descent.
- Negative sampling to speed up the learning process.

DD2418 Language Engineering
6e: Glove

Johan Boye, KTH

Glove

In 2014, Pennington, Socher and Manning proposed *Glove* (GLObal word VEctors).

Glove directly *specifies* how similar different word vectors should be, by means of a loss function that should be minimized

- similarity = dot product ($u \cdot v$)

Glove

Each word has two vectors:

- w (the focus word vector),
- $\widetilde{w}$ (the context word vector).

The dimensionality of the vectors is a hyperparameter of the method (typically 50-300).

We now compute for every pair of words i, j:

$$X_{ij} = \text{number of times word } j \text{ appears in the context of word } i$$

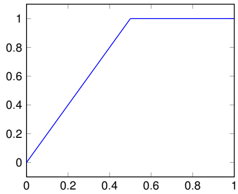
"Context" typically is a window ± 2 positions in the text.

Glove loss function

We then minimize the following loss function:

$$\frac{1}{2}\sum_{i,j=1}^{V} f(X_{ij})(w_i^T \widetilde{w}_j - \log X_{ij})^2$$

where $f(x)$ is a function that penalizes very common word combinations, e.g.: $f(x) = \begin{cases} (x/x_{max})^{0.75} & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases}$



Glove training

Use stochastic gradient descent. Loss function for a given pair of words i, j:

$$L_{ij} = \frac{1}{2} f(X_{ij})(w_i^T \widetilde{w}_j - \log X_{ij})^2$$

Training regime (short version):

1. In each iteration, randomly select i, j such that $X_{ij} > 0$.
2. Compute the gradient of L_{ij} w.r.t. w_i and $\widetilde{w}_j$
3. Update w_i and $\widetilde{w}_j$ using the gradients
4. Keep repeating from (1) until the the overall loss L increases.

Glove training

Training regime (longer version):

1. In each iteration, randomly select i, j such that $X_{ij} > 0$.
   - **Comment:** Choose i according to how common it is, e.g. how many words it co-appears with.
2. Compute the gradient of L w.r.t. w_i and $\widetilde{w}_j$
3. Update w_i and $\widetilde{w}_j$ using the gradients
   - **Comment:** Start with learning rate 0.05. Decrease it slightly every 1,000,000 iterations.
4. Keep repeating from (1) until the the overall loss L increases.
   - **Comment:** Calculate the loss every 100,000 iterations. If the loss increases more than 5 measurements in a row, terminate

Glove end result

Manning et al. recommends: $w + \widetilde{w}$ is the final word vector for the word.

However, in assignment 3 we are going to consider w as the final word vector.

Pretrained Glove vectors

Pre-trained Glove vectors for English, trained on Wikipedia, (with dimensionalities 50, 100, 200, 300) can be downloaded from `nlp.stanford.edu/data/glove.6B.zip`

There are also bigger collections, trained on Common Crawl, for download.

Sum-up

Computational lexical semantics: Representing the meaning of words

- Distributional hypothesis
- Distributed word representation (as a vector of numbers)
- Latent Semantic Analysis (1988)
- Random Indexing (2000)
- word2vec (2013)
- Glove (2014)
- FastText (2016)