Word Embedding: word2vec

Jianguo Lu

October 21, 2024

Overview

Word embedding

- A set of language modeling and feature learning techniques in natural language processing (NLP)
- Words are mapped to short and dense vectors of real numbers.
- Coined by Yoshua Bengio in 2003
  - Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model.
    Journal of machine learning research, 3(Feb):1137–1155, 2003
- Popularized by word2vec
  - Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean.

    Distributed representations of words and phrases and their compositionality.
    In *NIPS*, pages 3111–3119, 2013
  - Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space.
    arXiv preprint arXiv:1301.3781, 2013
- Related terms:
  - Distributional semantic model (in computational linguistics)
  - Distributed representation

Why word embedding?

- Useful by itself (e.g., calculating the most similar words)
- Essential for semantic search
- For downstream NLP tasks (e.g., classification)
- Starting point for other embeddings, e.g., graph embedding, molecule embedding, drug embedding, ...

Why called "embedding"

- In math, *embedding* is an injective map from one mathematical structure to another that preserves certain properties or structures.
  - distinct words are mapped to distinct vectors

$$f(x) = f(y) \Longrightarrow x = y$$

- in CS, *embedding* is often a map from from high-dimensional data into a lower-dimensional, continuous vector space.
- *placing* the sentence in a geometric space where relationships between sentences (or words) can be meaningfully represented by distances or directions.
- Requirement for good word/sentence embedding:
  - similar sentences (in meaning) will have similar embeddings
  - while dissimilar sentences will have more distant embeddings.

why called "distributed" representation

- the information (semantic, syntactic, etc.) about an a word (or sentence) is distributed across many dimensions of a vector,
- with each dimension capturing different aspects of the meaning.
- in contrast to one-dimensional (local) representations.

local	[0, 0, 1, 0, 0]
distributed	[0.25, 0.1, -0.5, 0.7, 0.3],

Distributional representation

Harris, Z. (1954)

"words that are used and occur in the same contexts tend to purport similar meanings."

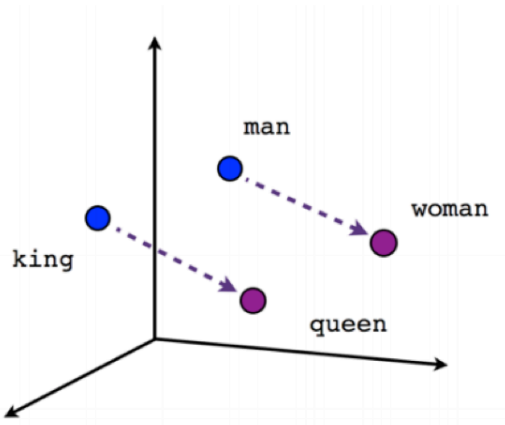
Firth, J.R. (1957)

"a word is characterized by the company it keeps".

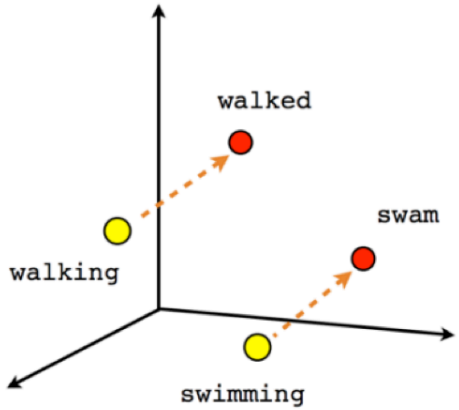
- Derived from the semantic theory of language usage
- You can get a lot of value by representing a word by means of its neighbors
- One of the most successful ideas of modern statistical NLP

...	government	debt	problems	turning	into	**banking**	crises	as	has	happened	
...	saying	that	Europe	needs	unified	**banking**	regulation	to	replace	the	ho

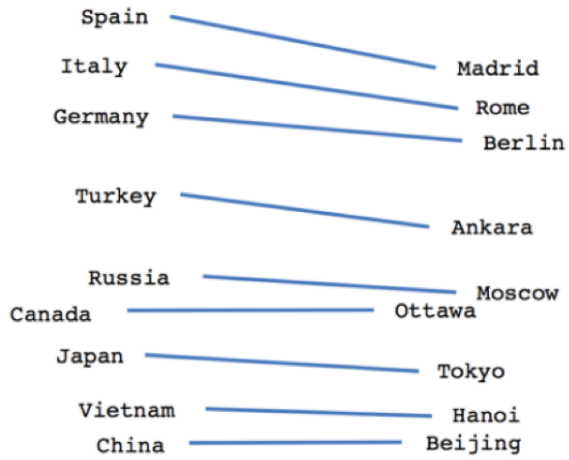
Examples of word embedding results



Male-Female



Verb tense



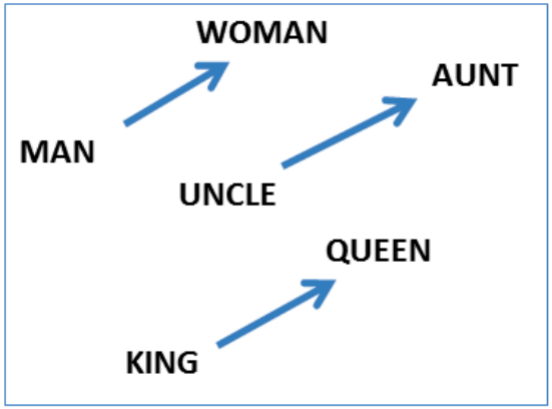
Country-Capital

Answer analogical questions, e.g

$$Man : Woman = King :?$$

The answer will be Queen.

Dimensions of representation



Vector representation

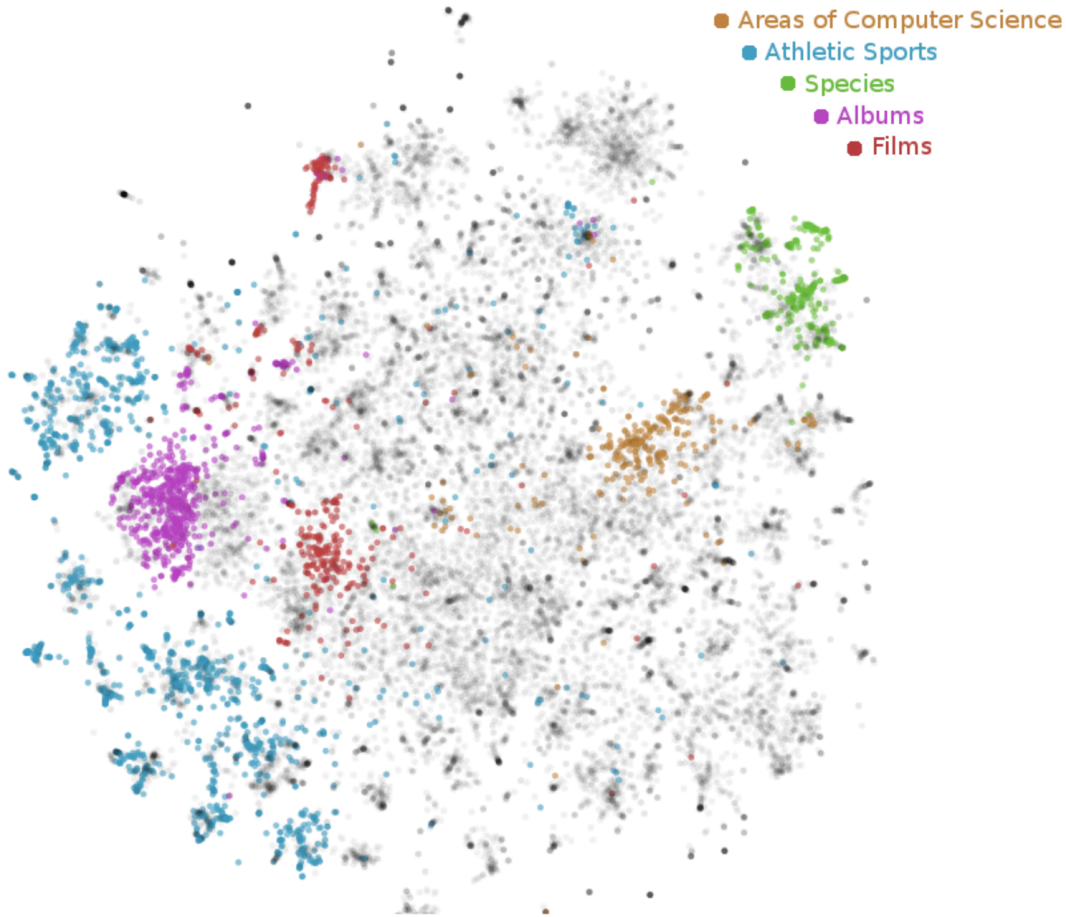
Vector representation of words is not new:

- Vector space model (TFIDF etc).
- LDA (Latent Dirichlet Allocation, for topic modelling)
  - David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation.
    Journal of machine Learning research, 3(Jan):993–1022, 2003
- SVD, PCA, PPMI +SVD
- LSA Latent Semantic Analysis
- Word2Vec, GloVe
- ...

Some approaches are correlated. e.g., word2vec and SVD+PPMI are mathematically related (almost equivalent).

- Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization.
  In *Advances in neural information processing systems*, pages 2177–2185, 2014

Beyond word embedding: document/sentence/paragraph embedding



- Visualization of Wikipedia paragraph vectors using t-SNE.
- From Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents.
  In *ICML*, volume 14, pages 1188–1196, 2014

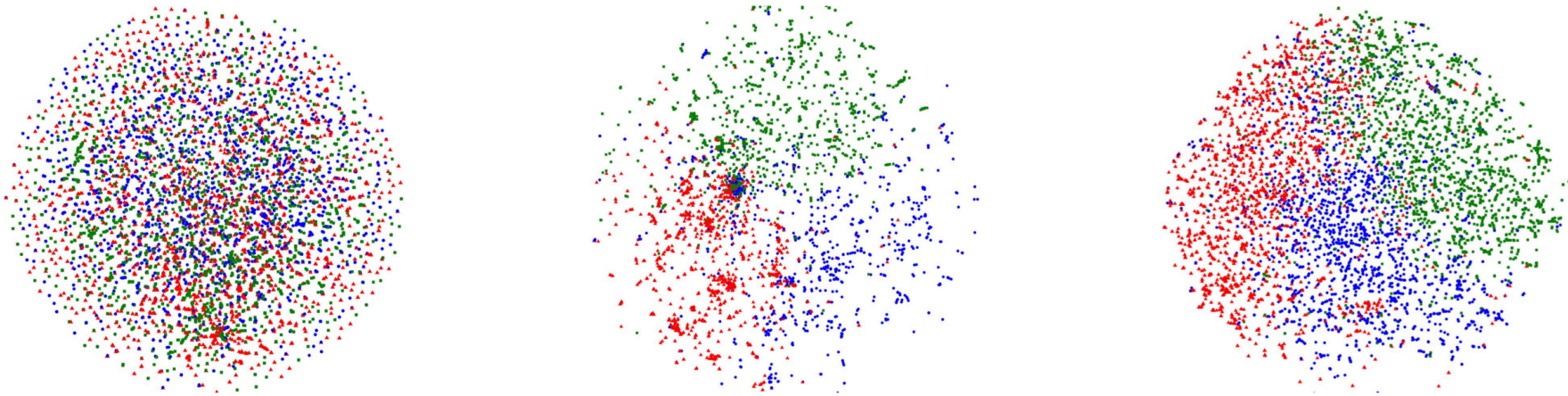
Contextual embedding

- Non-contextual
  - Embedding of a word is unique regardless of its context
  - e.g. embeddings of ‘bank‘ in the following two sentences are the same

    government debt problems turning into banking crises ...
    We sat on the river bank, watching the water flow ...

  - Also called static embedding
  - Approaches for static embedding: Word2Vec, Glove
- Contextual Embedding
  - Also called dynamic embedding
  - Embedding of a word is generated dynamically depending on the context of the word
  - Examples: BERT, LLAMA, LLAMA, ChatGPT ...

Network embedding



(a) GF (b) DeepWalk (c) LINE(2nd)

Figure: From Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, pages 1067–1077. ACM, 2015

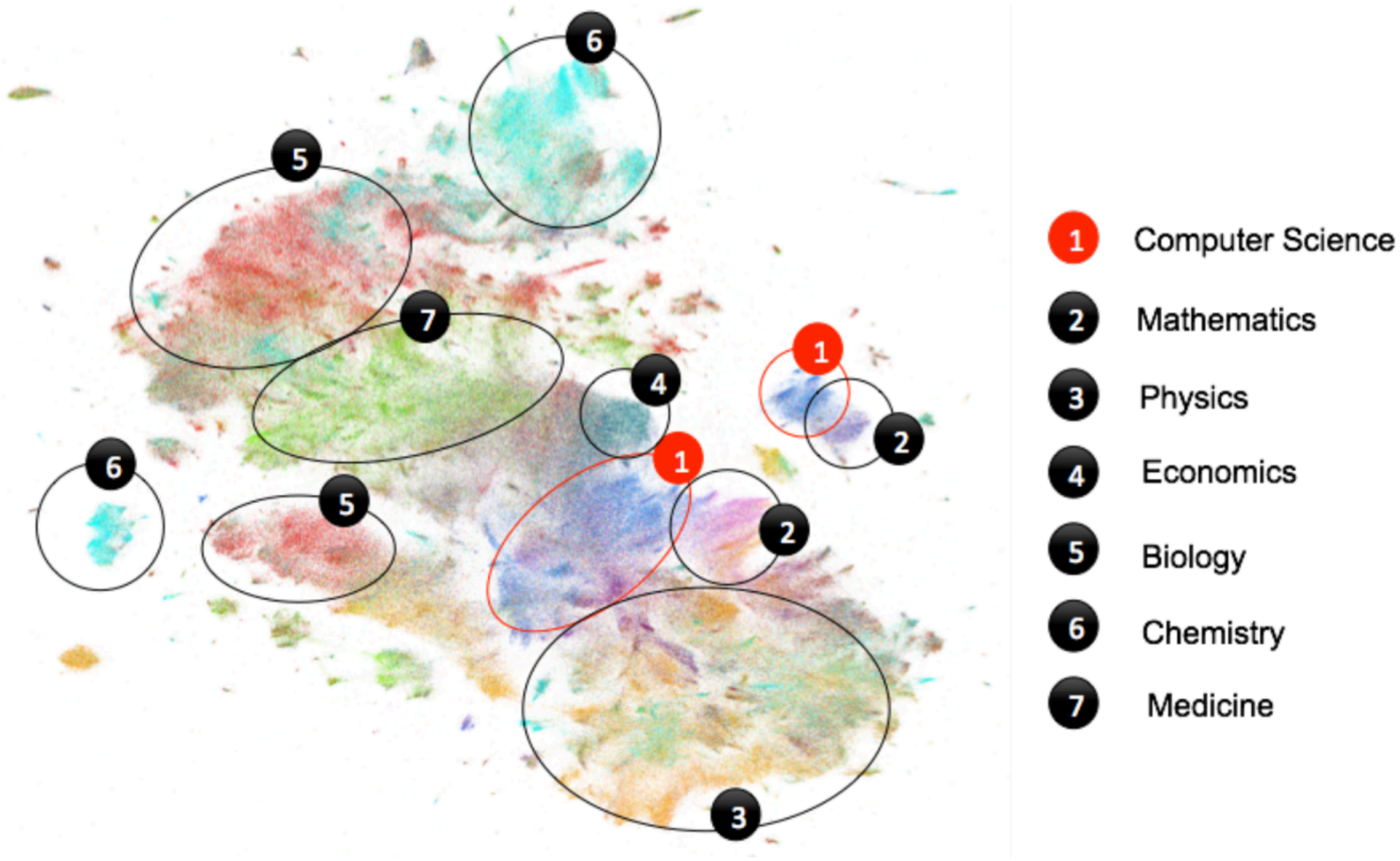
- Visualization of the co-author network.
- The authors are mapped to the 2-D space using the t-SNE package with learned embeddings as input.
- Color of a node indicates the community of the author.
- Red: "data Mining," blue: "machine learning," green: "computer vision."

More representative works for network embedding

- DeepWalk: Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations.
  In *SIGKDD*, pages 701–710. ACM, 2014
- Node2Vec: Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks.
  In *SIGKDD*, pages 855–864. ACM, 2016
- Graph Convolution Networks

Network embedding for 6M authors

From KDD 2017 Tutorial, By Tang et al.


6
5
7
4
1
2
6
1
5
2
3
1 Computer Science
2 Mathematics
3 Physics
4 Economics
5 Biology
6 Chemistry
7 Medicine

Paper (linked data) embedding

- Suhang Wang, Jiliang Tang, Charu Aggarwal, and Huan Liu. Linked document embedding for classification.
  In *CIKM*, pages 115–124. ACM, 2016
- Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang.

  Network representation learning with rich text information.
  In *IJCAI*, pages 2111–2117, 2015
- Soumyajit Ganguly and Vikram Pudi. Paper2vec: Combining graph and text information for scientific paper representation.
  2017
- Jian Tang, Meng Qu, and Qiaozhu Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks.
  In *SIGKDD*, pages 1165–1174. ACM, 2015

Meaning of a word

Definition: Meaning(Webster dictionary)

- the thing one intends to convey especially by language

Meaning from WordNet

- Semantically oriented dictionary
- similar to a thesaurus, with richer structure
- 155,287 English words, 117,659 synonyms
- nltk in python (and other languages) support WordNet

```
>>>from nltk.corpus import wordnet as wn
>>> wn.synsets('car')[0].lemma_names()
[u'car', u'auto', u'automobile', u'machine', u'motorcar']

>>> wn.synsets('car')[1].lemma_names()
[u'car', u'railcar', u'railway_car', u'railroad_car']

>>> wn.synsets('car')[2].lemma_names()
[u'car', u'gondola']

>>> wn.synsets('car')[3].lemma_names()
[u'car', u'elevator_car']
>>>
```

It is similar to a dictionary

car noun
\ kär , dialectal also ȯkr , kyär \
Definition of Car (Entry 1 of 2)
1 : a vehicle moving on wheels: such as
a archaic : CARRIAGE, CHARIOT
b : a vehicle designed to move on rails (as of a railroad)
The train has 20 cars.
c : AUTOMOBILE
traveled to Boston by car
2 : the passenger compartment of an elevator
3 : the part of an airship or balloon that carries the passeng

WordNet has synonyms, hypernyms, etc.

```
>>> panda=wn.synset('panda.n.01')
>>> hyper=lambda s:s.hypernyms()
>>> list(panda.closure(hyper))
[Synset('procyonid.n.01'), Synset('carnivore.n.01'),
Synset('placental.n.01'), Synset('mammal.n.01'),
Synset('vertebrate.n.01'), Synset('chordate.n.01'),
Synset('animal.n.01'), Synset('organism.n.01'),
Synset('living_thing.n.01'), Synset('whole.n.02'),
Synset('object.n.01'), Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

Problems with WordNet

- Miss subtle differences,
  - e.g., synonyms: adept, expert, good, practiced, proficient, skillful?
- Miss new words. Impossible to keep up to date especially in online social networks such as Twitter:
  - e.g. looook, l00k, helloooo, ninja
- Subjective
- Requires human labor to create and adapt
- Hard to compute accurate word similarity
  - How to give a real number between 0 to 1 to measure the similarity?

Similar words from word2vec

```
>>> model.most_similar('car', topn=29)
[('vehicle', 0.7560694813728333), ('truck', 0.6907597184181213),
('cars', 0.6613855361938477), ('bicycle', 0.660092830657959),
('vehicles', 0.65554547309875 49), ('door', 0.6354320645332336),
('ship', 0.6315749883651733), ('trailer', 0.6131559610366821),
('seat', 0.6075621843338013), ('aircrafts', 0.6072666049003601),
('driving', 0.606517612934 1125),
```

- Training data is `dblp_title.txt`.
- What is better is the muiti-dimensional representation

Compare with the similarity from WordNet

```
synonym of 'good'
S: (adj) full, good
S: (adj) estimable, good, honorable, respectable
S: (adj) beneficial, good
S: (adj) good, just, upright
S: (adj) adept, expert, good, practiced, proficient, skillful
S: (adj) dear, good, near
S: (adj) good, right, ripe

S: (adv) well, good
S: (adv) thoroughly, soundly, good
S: (n) good, goodness
S: (n) commodity, trade good, good
```

"one-hot" representation

- It is a localist representation
- words are regarded as atomic symbols
- This is a vector with one 1 and a lot of zeroes

$$
\begin{aligned}
Hotel^T &= [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0] \\
Conference^T &= [0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0] \\
Windsor^T &= [0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]
\end{aligned} \tag{1}
$$

Problems:

- Dimensionality of the vector will be the size of vocabulary. e.g. 13 M for Google 1T.
- $Hotel^T Conference = 0$

How to make neighbours represent words?

- Answer: With a co-occurrence matrix X
- 2 options: full document vs windows
  - Word - document cooccurrence matrix will give general topics (all sports terms will have similar entries) leading to Latent Semantic Analysis.
  - Instead: Window around each word to captures both syntactic (POS) and semantic information

...	government	debt	problems	turning	into	**banking**	crises	as	has	happened

Word-word Co-occurrence matrix

- Co-occurrence can be interpreted as an indicator of semantic proximity of words.
- colab link

Silence is the language of God, all else is poor translation.
Rumi (1207,1273)

counts	silence	is	the	language	of	God
silince	0	1	0	0	0	0
is	1	0	1	0	0	0
...						

Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography.
Computational linguistics, 16(1):22–29, 1990

Basic idea of learning neural network word embeddings

a model that aims to predict between a center word w_t and context words in terms of word vectors

$$p(context|w_t) = \ldots$$

which has a loss function, e.g.,

$$J = 1 - p(context|w_t)$$

- Look at many positions t in a big language corpus
- Keep adjusting the vector representations of words to minimize this loss

Main idea of word2vec

Predict between every word and its context words!
Two algorithms

- Skip-grams (SG): Predict context words given target (position independent)
- Continuous Bag of Words (CBOW): Predict target word from bag-of-words context

Two (moderately efficient) training methods

- Hierarchical softmax
- Negative sampling

The winner combination: SG+NS, i.e., SGNS.

word2vec (Skip-gram) as an optimization problem

- For each word $t = 1, \ldots, T$, predict surrounding words in a window of "radius" m of every word.
- Objective function: Maximize the probability of any context word given the current center word:

$$J(\theta) = \prod_{t=1}^{T} \prod_{-m \leq j \leq m \wedge j \neq 0} P(w_{t+j} | w_t; \theta) \tag{2}$$

- Negative log likelihood:

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m \wedge j \neq 0} \log(P(w_{t+j} | w_t; \theta)) \tag{3}$$

  θ represents all variables we will optimize
- Skip-gram uses softmax for $P(w_i | w_t)$:

$$P(w_i \mid w_t) = \frac{\exp(\mathbf{v}_{w_t} \cdot \mathbf{v}'_{w_i})}{\sum_{j=1}^{V} \exp(\mathbf{v}_{w_t} \cdot \mathbf{v}'_{w_j})} \tag{4}$$

Idea 1: Reduce the dimensionality

Singular Value Decomposition (SVD)

$$X = U\Sigma V^T \tag{5}$$

- The columns of U contain the eigenvectors of XX^T .
- The columns of V contain the eigenvectors of XT^X .
- Σ is diagonal matrix. Diagonal values are eigenvalues of XX^T or X^TX.
- Computational cost in SVD scales quadratically for $d \times n$ matrix: $O(nd^2)$ (when $d < n$)
- Not possible for large number of words or document.

Idea 2: Learn low-dimensional vectors directly

- Learning representations by back-propagating errors. (Rumelhnrt, Hinton, Williams 1986)
- A Neural Probabilistic Language Model (Bengio et al., 2003)
- Natural Language Processing (Almost) from Scratch (Collobert et al., 2011)
- Efficient Estimation of Word Representations in Vector Space (Mikolov et al., 2013)
- GloVe: Global Vectors for Word Representation (Pennington et al., 2014)

Idea 2

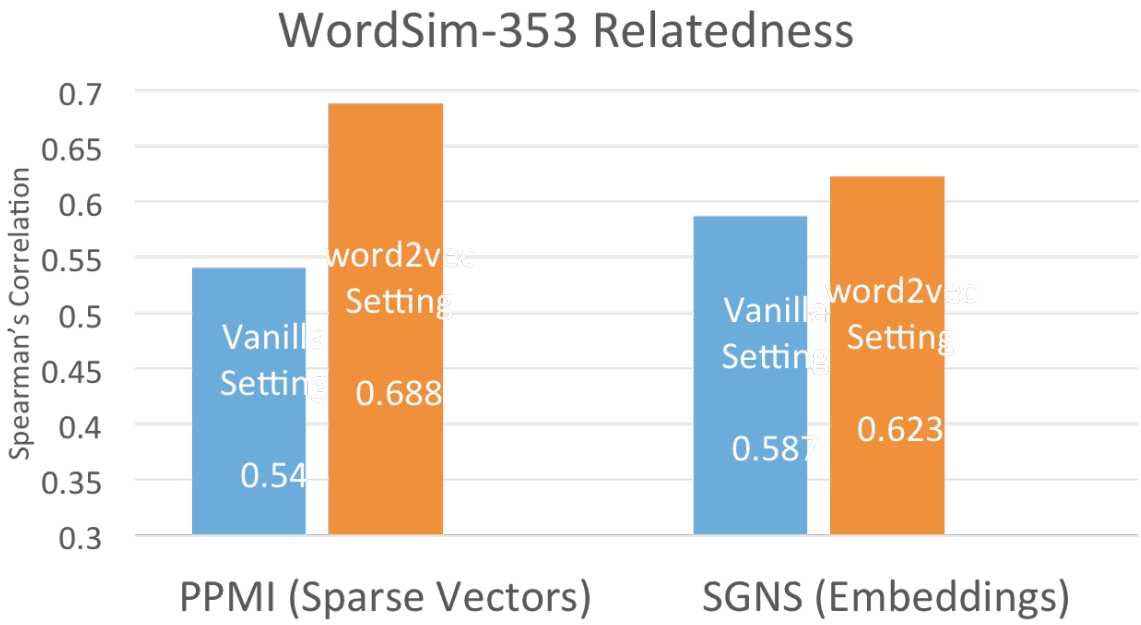
- Predict surrounding words of every word(word2rec)
  - Update the weight every time two words co-occur
  - The same co-occurrence pair (w_i, w_j) may appear many times
  - It is not efficient to update the weight multiple times
- Capture co-occurrence counts directly (Glove)
- They are fast and can incorporate a new sentence/document or add a word to the vocabulary.

Word2vec has a lot of algorithms and hyper parameters

- 9+ Hyperparameters
- 4 +Word Representation Algorithms
  - PPMI (Sparse and Explicit) SVD(PPMI)
  - SGNS
  - GloVe
- 8+ Benchmarks (6 Word Similarity Tasks, 2 Analogy Tasks)

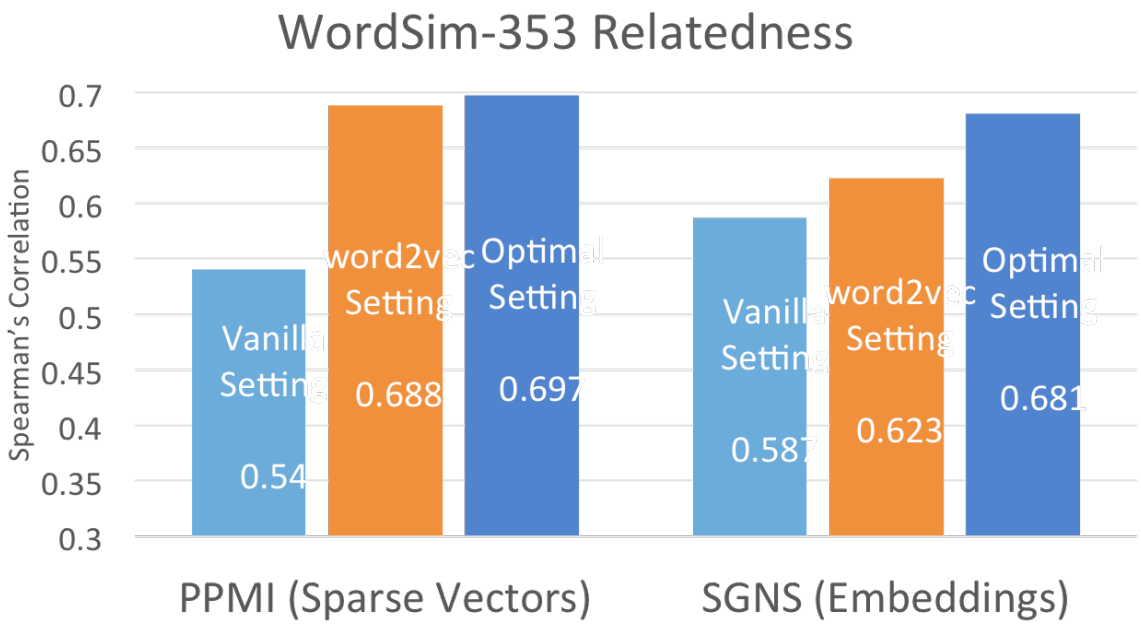
Text pre-processing

- lower-casing (yes)
- stemming. (e.g., query vs. queries) (yes)
- Bigrams/phrases



(From slides by Omer Levy et al.)

- PPMI vs SGNS
- Hyperparameters often have stronger effects than algorithms
- Hyperparameters often have stronger effects than more data
- Prior superiority claims were not accurate


WordSim-353 Relatedness
Spearman's Correlation
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
Vanilla Setting
0.54
word2vec Setting
0.688
Optimal Setting
0.697
Vanilla Setting
0.587
word2vec Setting
0.623
Optimal Setting
0.681
PPMI (Sparse Vectors)
SGNS (Embeddings)

word2vec implementations

- Mikolov's original code in C
- Gensim word2vec in Python
- DL4J in java `https://deeplearning4j.org/`
- ...

word2vec in C

- Original site at Google (has deadlinks): `https://code.google.com/archive/p/word2vec/`
- Mirror at GitHub: `https://github.com/svn2github/word2vec`
- Detailed comments: `https://github.com/chrisjmccormick/word2vec_commented`

Run the code

```
$ make
$ ./demo-word.sh
$ ./distance vectors.bin

Enter word or sentence (EXIT to break): cat

Word: cat  Position in vocabulary: 2601
```

Word	Cosine distance
meow	0.621209
cats	0.568651
feline	0.550209
caracal	0.542168
dog	0.538465

- vectors.bin is the vector representation of the words
- Other scripts for demo, e.g., ./demo-phrases.sh

gensim implementation for word2vec

- Open source Python lib for NLP
- Focus on topic modelling, latent semantic modelling.
- Developed by Radim Rehurek
- PhD in 2011: `https://radimrehurek.com/phd_rehurek.pdf`
- Benefits of using Gensim implementation:
  - come with other embedding lib such as LDA and LSI.
  - Python has handy plot lib
- Install gensim

```
pip install --upgrade gensim
```

Starter code

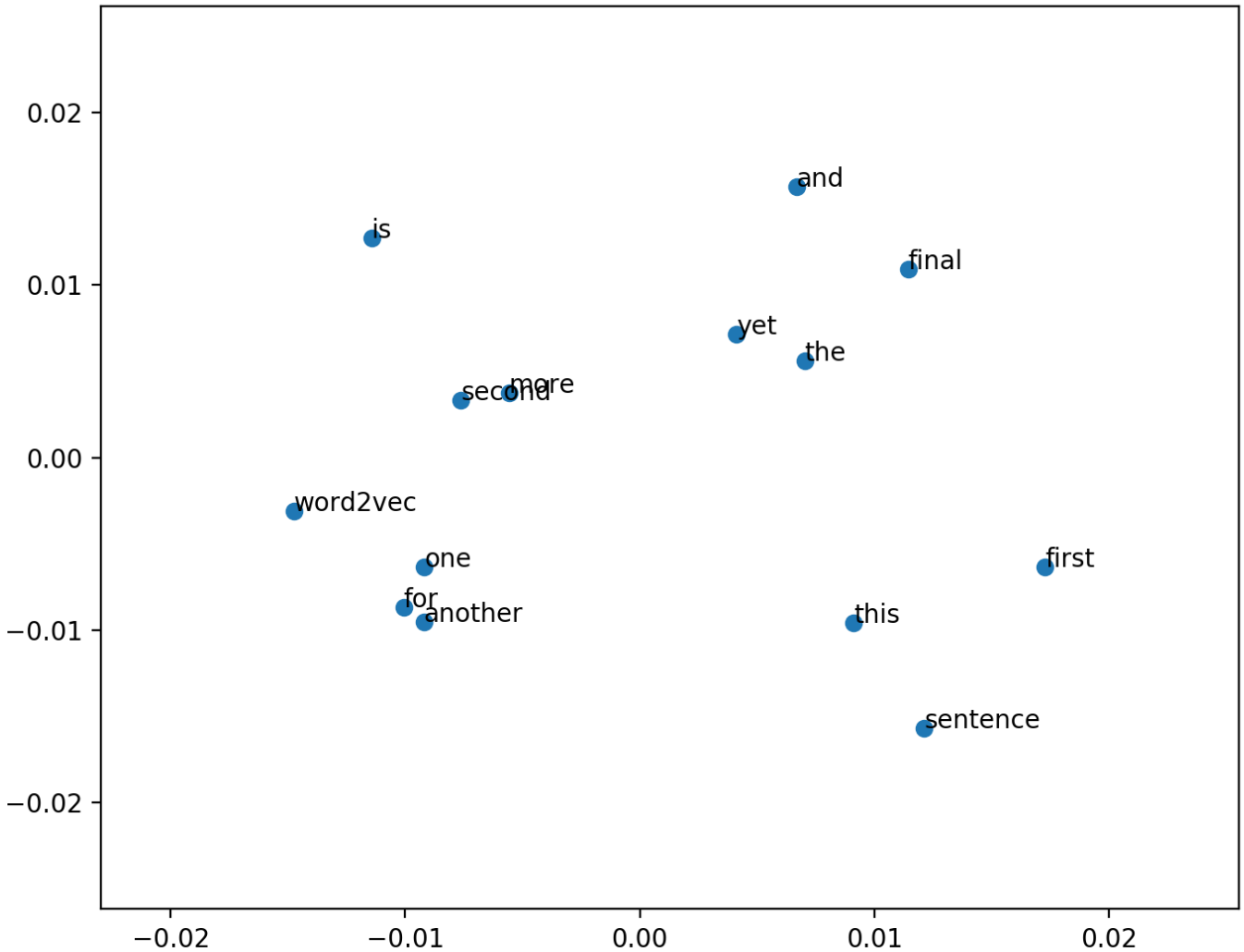
```
from gensim.models import Word2Vec
from sklearn.decomposition import PCA
from matplotlib import pyplot
# define training data
sentences = [['this', 'is', 'the', 'first', 'sentence', 'for', 'word2vec'],
    ['this', 'is', 'the', 'second', 'sentence'],
    ['yet', 'another', 'sentence'],
    ['one', 'more', 'sentence'],
    ['and', 'the', 'final', 'sentence']]
# train model
model = Word2Vec(sentences, min_count=1)
# fit a 2d PCA model to the vectors
X = model[model.wv.vocab]
pca = PCA(n_components=2)
result = pca.fit_transform(X)
# create a scatter plot of the projection
pyplot.scatter(result[:, 0], result[:, 1])
words = list(model.wv.vocab)
for i, word in enumerate(words):
    pyplot.annotate(word, xy=(result[i, 0], result[i, 1]))
pyplot.show()
```

What is the 'model'?

```
>>> model['first']
array([ 0.14996897, -0.20207308,  0.3628332 ,  0.48634875, -0.9683252 ,
       -0.56452739,  0.49738097, -0.24710093,  0.90575856,  1.16950583,
        0.12466316, -0.23972373,  0.22282168, -0.12682317, -0.44225532,
       -0.09795734,  0.39110288, -0.40137786,  0.27168629, -0.10275133,
       -0.02124002,  0.30650523, -0.11591583, -0.46616486, -0.51625609,
       -0.1998333 , -0.0062433 , -0.43187553, -0.39892   , -0.36950222,
       ...
```

- The length of the array is dictated by the hyper-parameter `size`
- Normally a few hundred
- The default is 100 in Gensim implementation.
- Note that it is not normalized (to 1).

Visualization using PCA



Logging the process

```
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)

Jianguos-MBP:mycode jianguolu$ 2017-11-05 09:41:26,372 : INFO : collecting all words and their counts
2017-11-05 09:41:26,372 : INFO : PROGRESS: at sentence #0, processed 0 words, keeping 0 word types
2017-11-05 09:41:26,372 : INFO : collected 14 word types from a corpus of 22 raw words and 5 sentences
2017-11-05 09:41:26,372 : INFO : Loading a fresh vocabulary
2017-11-05 09:41:26,372 : INFO : min_count=1 retains 14 unique words (100% of original 14, drops 0)
2017-11-05 09:41:26,372 : INFO : min_count=1 leaves 22 word corpus (100% of original 22, drops 0)
2017-11-05 09:41:26,373 : INFO : deleting the raw counts dictionary of 14 items
2017-11-05 09:41:26,373 : INFO : sample=0.001 downsamples 14 most-common words
2017-11-05 09:41:26,373 : INFO : downsampling leaves estimated 2 word corpus (12.7% of prior 22)
2017-11-05 09:41:26,373 : INFO : estimated required memory for 14 words and 100 dimensions: 18200 bytes
2017-11-05 09:41:26,373 : INFO : resetting layer weights
2017-11-05 09:41:26,373 : INFO : training model with 3 workers on 14 vocabulary and 100 features,
   using sg=0 hs=0 sample=0.001 negative=5 window=5
2017-11-05 09:41:26,373 : INFO : expecting 5 sentences,
   matching count from corpus used for vocabulary survey
2017-11-05 09:41:26,383 : INFO : worker thread finished; awaiting finish of 2 more threads
2017-11-05 09:41:26,384 : INFO : worker thread finished; awaiting finish of 1 more threads
2017-11-05 09:41:26,384 : INFO : worker thread finished; awaiting finish of 0 more threads
2017-11-05 09:41:26,384 : INFO : training on 22000 raw words (2811 effective words) took 0.0s,
  292473 effective words/s
```

Hyper-parameter 1: Subsampling

Each word w_i in the training set is discarded with probability computed by the formula

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{6}$$

- $f(w_i)$ is the frequency of the word (divided by the corpus size), t is the pre-chosen threshold value.
- The default one in Gensim is t=0.001.
- Mikolov suggests t=0.00001.
- Frequent words have higher probability being discarded.

```
2017-11-05 13:32:05,470 : INFO : sample=0 downsamples 0 most-common words
2017-11-05 13:32:05,471 : INFO : downsampling leaves estimated 22 word corpus (100.0% of prior 22)
```

Read from a file and SIGMOD data

```
import gensim
PATH='/Users/jianguolu/data/'
txtfile= open(PATH+'sigmod_title.txt','r')
sentences=[line.lower().strip().split(' ') for line in txtfile.readlines()]
model = gensim.models.Word2Vec(sentences, min_count=2, iter=5)
test='query'
print 'words similar to \''+ test + '\':\t'+ str(model.most_similar(test))

words similar to 'query':
[('with', 0.9999337196350098), ('in', 0.9999307990074158),
('and', 0.9999281764030457), ('for', 0.9999271631240845),
('queries', 0.9999268054962158), ('a', 0.9999261498451233),
('based', 0.9999228715896606), ('databases', 0.999917626380920 4),
('using', 0.9999162554740906), ('efficient', 0.999914526939392 1)]
```

Similar words run another time:

```
words similar to 'query':
[('queries', 0.9999304413795471), ('with', 0.9999302625656128),
('a', 0.9999301433563232), ('and', 0.9999282360076904),
('for', 0.999927818775177), ('in', 0.9999264478683472),
('based', 0.9999209046363831), ('databases', 0.999919056892395),
('using', 0.9999158382415771), ('to', 0.999911129474639 9)]
```

- The result is not good
- Similarities are close to one. (why?)
- Outputs vary from run to run (why?)

Hyper-parameter 2: learning rate alpha

```
model = gensim.models.Word2Vec(sentenceList, alpha=0.3,iter=5)

words similar to 'query':
[('reducing', 0.5237733125686646), ('analytic', 0.5013805627822876),
('algebra', 0.4918916225433496), ('plans', 0.4694745540618965),
('numeric', 0.4621663689613342), ('portable', 0.4466435313224792),
('natural', 0.4325773417949676), ('output', 0.4266804456710815),
('recursive', 0.4199021160602569), ('class', 0.4170923531055450),
('mapreduce', 0.4162992537021637), ('expressions', 0.4108825922012329),
```

- Result improves, still not good.
- Learning rate decreases during the training process (code in C):

```
alpha = starting_alpha*(1-word_count_actual/(real)(iter*train_words+1));
if (alpha < starting_alpha * 0.0001) alpha=starting_alpha*0.0001;
```

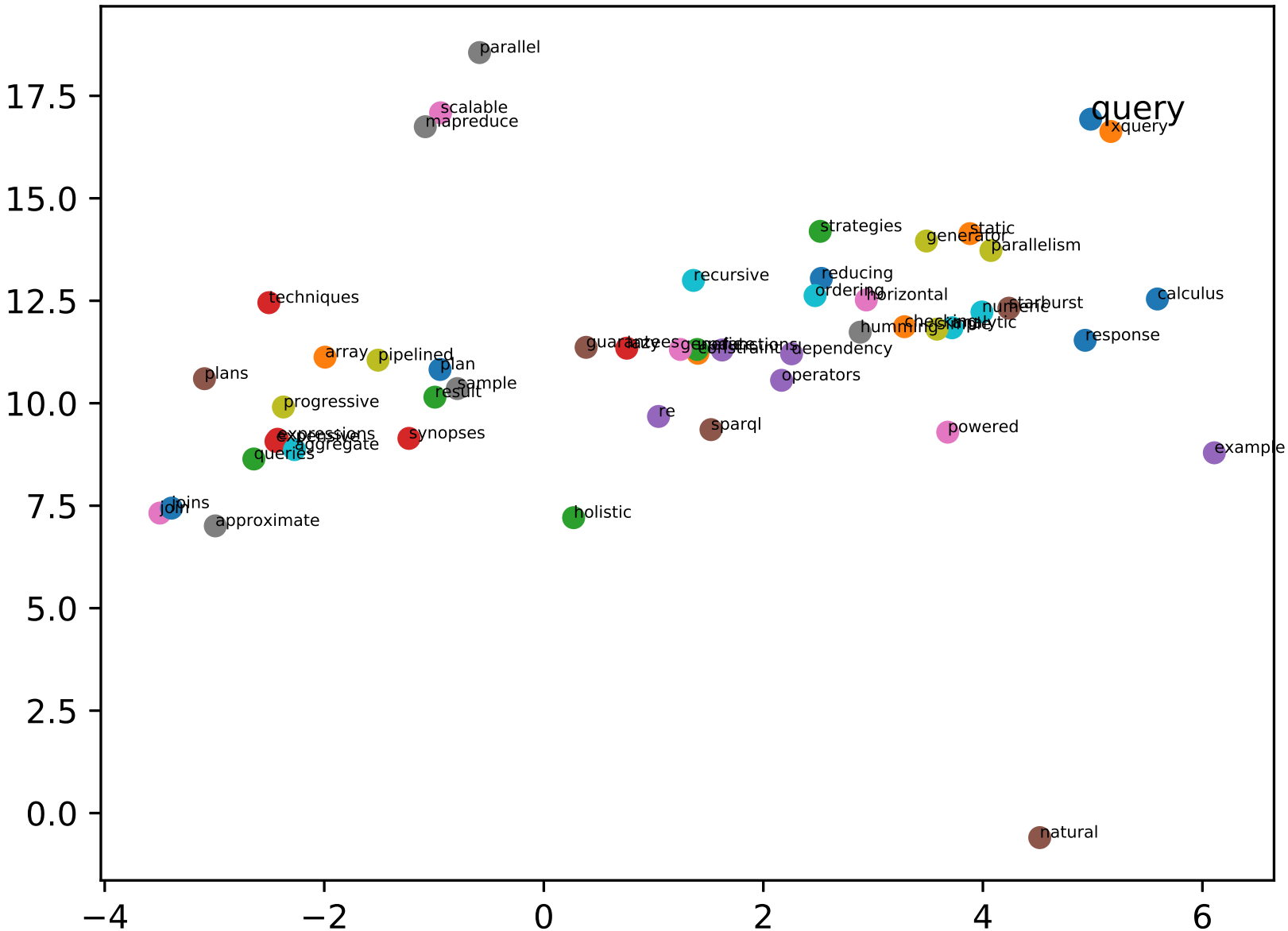
Hyper-parameter 3: iteration or epochs

```
model = gensim.models.Word2Vec(sentences, min_count=2, iter=50)
print str(model.most_similar('query'))

Jianguos-MBP:mycode jianguolu$ python w2v_sigmod.py
[('xquery', 0.6625971794128418),
('question', 0.6376067399978638),
('queries', 0.6186379194259644),
('natural', 0.6147118806838989),
('progressive', 0.6094042658805847),
('recursive', 0.6039171218872 07),
('grouping', 0.6024632453918457),
('lazy', 0.5983935594558716),
('rate', 0.596043586730957),
('update', 0.5865614414215088)]
```

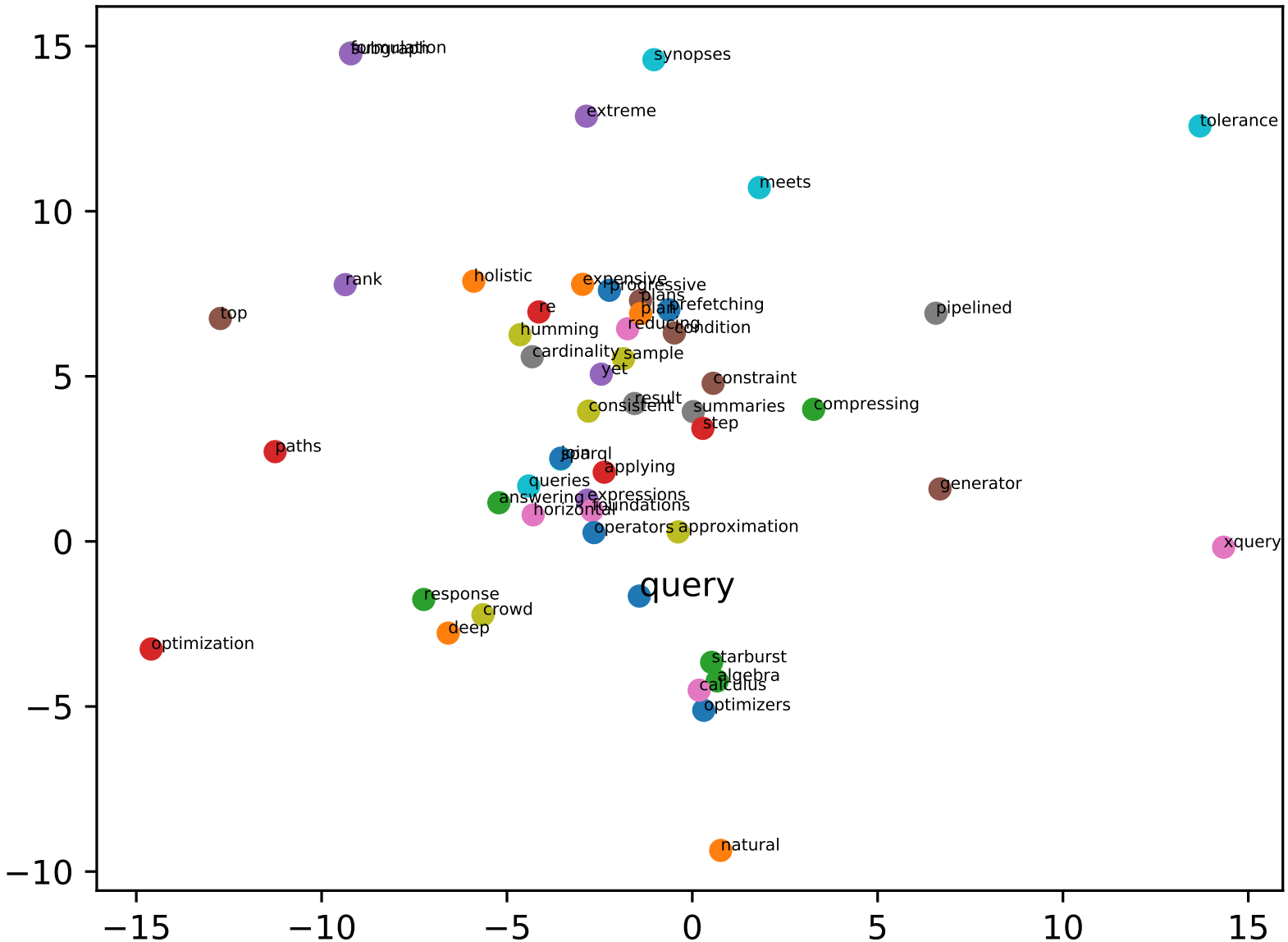
- Iteration is 50 now. The default is 5.

Embedding produced by CBOW



Embedding produced by SG

model = gensim.models.Word2Vec(sentences, sg=1, iter=50)



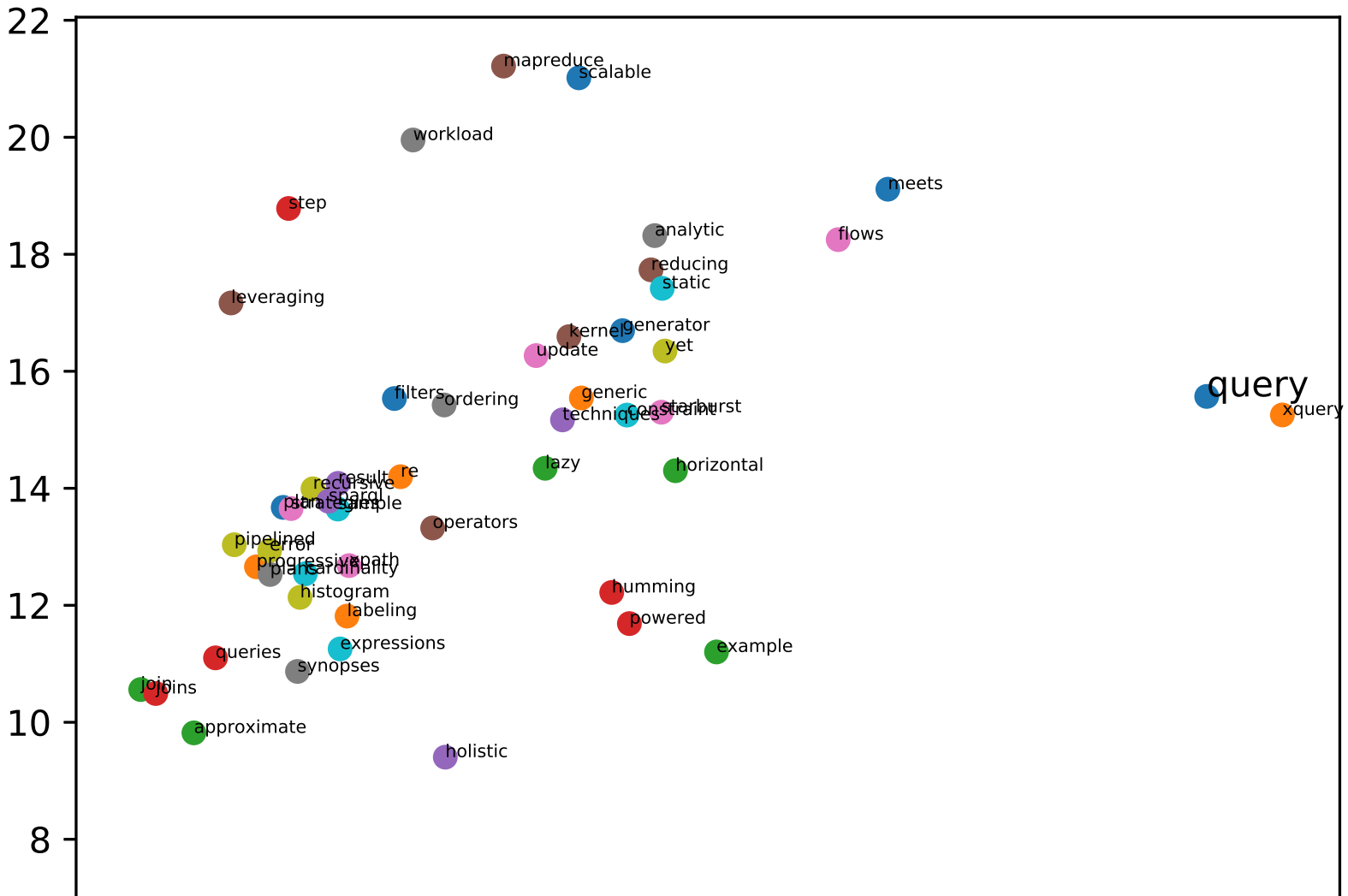
Hyper-parameter 4: min-count

- min_count: ignore all words with total frequency lower than this.
- words similar to 'query' when min_count=2:
- should decrease min_count when training data is small.

[('xquery', 0.63210129737854), ('lazy', 0.6053866147994995), ('queries', 0.5886513590812683), ('natural', 0.5693424940109253), ('operators', 0.5228461027145386), ('update', 0.5184661746025085), ('analytic', 0.515886902809143 1),

Visualization using t-SNE

- min_count =1
- note 'mapreduce' and 'scalable' (and 'workload') are close. 'join' and 'joins' are close.



Visualization using t-SNE

Introduced by van der Maaten and Hinton in 2008.
`https://lvdmaaten.github.io/tsne/`

- t-Distributed Stochastic Neighbor Embedding (t-SNE)
- Dimensionality reduction
- Particularly well suited for the visualization of high-dimensional datasets.
- Scalability is a problem
- A good tutorial: `https://distill.pub/2016/misread-tsne/`

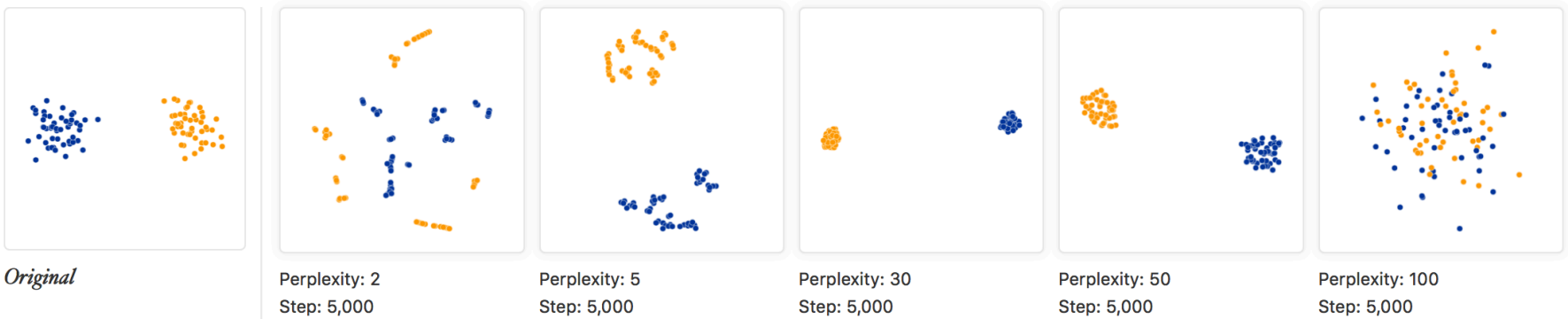
Starter code for t-SNE

```
import numpy as np
from sklearn.manifold import TSNE
....
voc= list(model.wv.vocab)
words=model.most_similar('query', topn=50);
j=voc.index('query')

tmodel = TSNE(n_components=2)
np.set_printoptions(suppress=True)
V2=tmodel.fit_transform(X)
pyplot.scatter(V2[j,0],V2[j,1])
pyplot.annotate('query', xy=(V2[j, 0], V2[j, 1]), fontsize=10)
for i, word in enumerate(words):
        j=voc.index(word[0])
        pyplot.scatter(V2[j,0],V2[j,1])
        pyplot.annotate(word[0], xy=(V2[j, 0], V2[j, 1]), fontsize=5)
pyplot.savefig('../538_2017/fig/w2v_sigmod1.eps', format='eps', dpi=300)
pyplot.show()
```

Parameters: perplexity

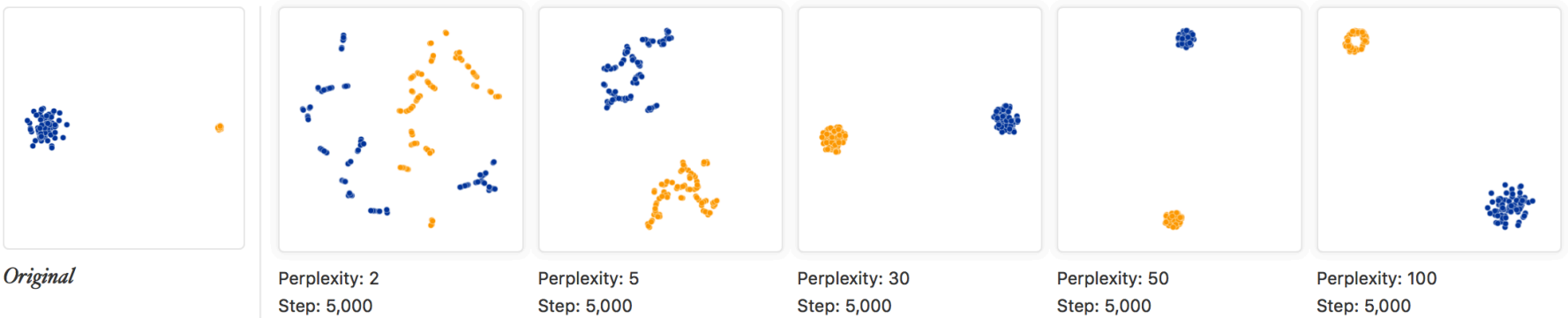
- The perplexity is related to the number of nearest neighbors that is used in other manifold learning algorithms.
- Larger datasets usually require a larger perplexity.
- Normally between 5 and 50.
- The perplexity should be smaller than the number of points



From `https://distill.pub/2016/misread-tsne/`

Cluster size/ Distance between nodes

- Expands dense clusters and contracts sparse ones
- Adapts 'distance' to regional density variation



Distance between clusters

- Distance between clusters increases with perplexity

Original

Perplexity: 2
Step: 5,000

Perplexity: 5
Step: 5,000

Perplexity: 30
Step: 5,000

Perplexity: 50
Step: 5,000

Perplexity: 100
Step: 5,000

Parameters of TSNE

- learning_rate : float, optional (default: 200.0)
- Usually in the range [10.0, 1000.0].
- If the learning rate is too high, the data may look like a ‘ball’ with any point approximately equidistant from its nearest neighbours.
- If the learning rate is too low, most points may look compressed in a dense cloud with few outliers.
- If the cost function gets stuck in a bad local minimum, increasing the learning rate may help.

DBLP data

5M papers.

```
Jianguos-MBP:data jianguolu$ wc dblp_title.txt
5459997 38065969 288463929 dblp_title.txt

words similar to 'query':
[('queries', 0.8232043385505676), ('queries.', 0.738012433052063),
('xquery', 0.6668544411659241), ('join', 0.6446130275726318),
('queries:', 0.62788987159729), ('sparql', 0.6264554858207703),
('xpath', 0.61639404296875), ('top-k', 0.6126927137374878),
('joins', 0.6064221858978271), ('question', 0.606041967868804 9)]
```

```
2017-11-05 16:10:03,033 : INFO : min_count=20 retains 50427 unique words
   (5% of original 941273, drops 890846)
2017-11-05 16:10:03,033 : INFO : min_count=20 leaves 36199226 word corpus
   (95% of original 38066117, drops 1866891)
2017-11-05 16:10:03,143 : INFO : deleting the raw counts dictionary of 941273 items
2017-11-05 16:10:03,194 : INFO : sample=0.001 downsamples 23 most-common words
2017-11-05 16:10:03,194 : INFO : downsampling leaves estimated 27252367 word corpus
   (75.3% of prior 36199226)
2017-11-05 16:10:03,194 : INFO : estimated required memory for 50427 words and
   100 dimensions: 65555100 bytes
2017-11-05 16:10:03,360 : INFO : resetting layer weights
2017-11-05 16:10:03,908 : INFO : training model with 3 workers on 50427
    vocabulary and 100 features, using sg=0 hs=0 sample=0.001 negative=5 window=5
```

Most similar words for 'testing'

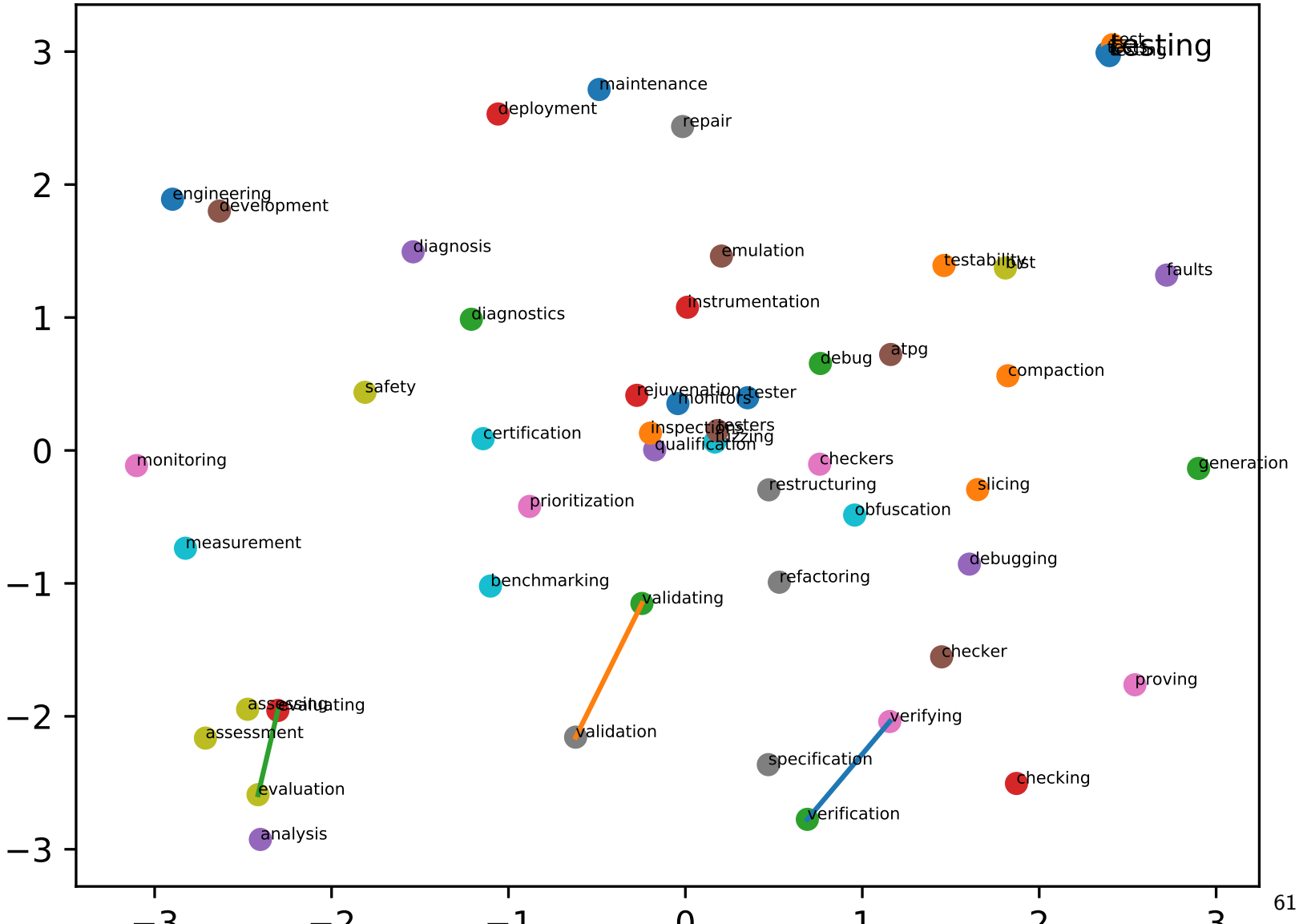
```
words similar to 'testing':
[('tests', 0.7413559556007385), ('test', 0.7291274070739746),
('verification', 0.6510030031204224), ('checking', 0.599650359153748),
('debugging', 0.5819920301437378), ('atpg', 0.5567741394042969),
('verifying', 0.5524342060089111), ('validation', 0.5462774038314819),
('bist', 0.532925367355346 7), ('certification', 0.5001979470252991)]
```

ATPG:

- Acronym for Automatic Test Pattern Generation.
- An electronic design automation method/technology used to find an input (or test) sequence.

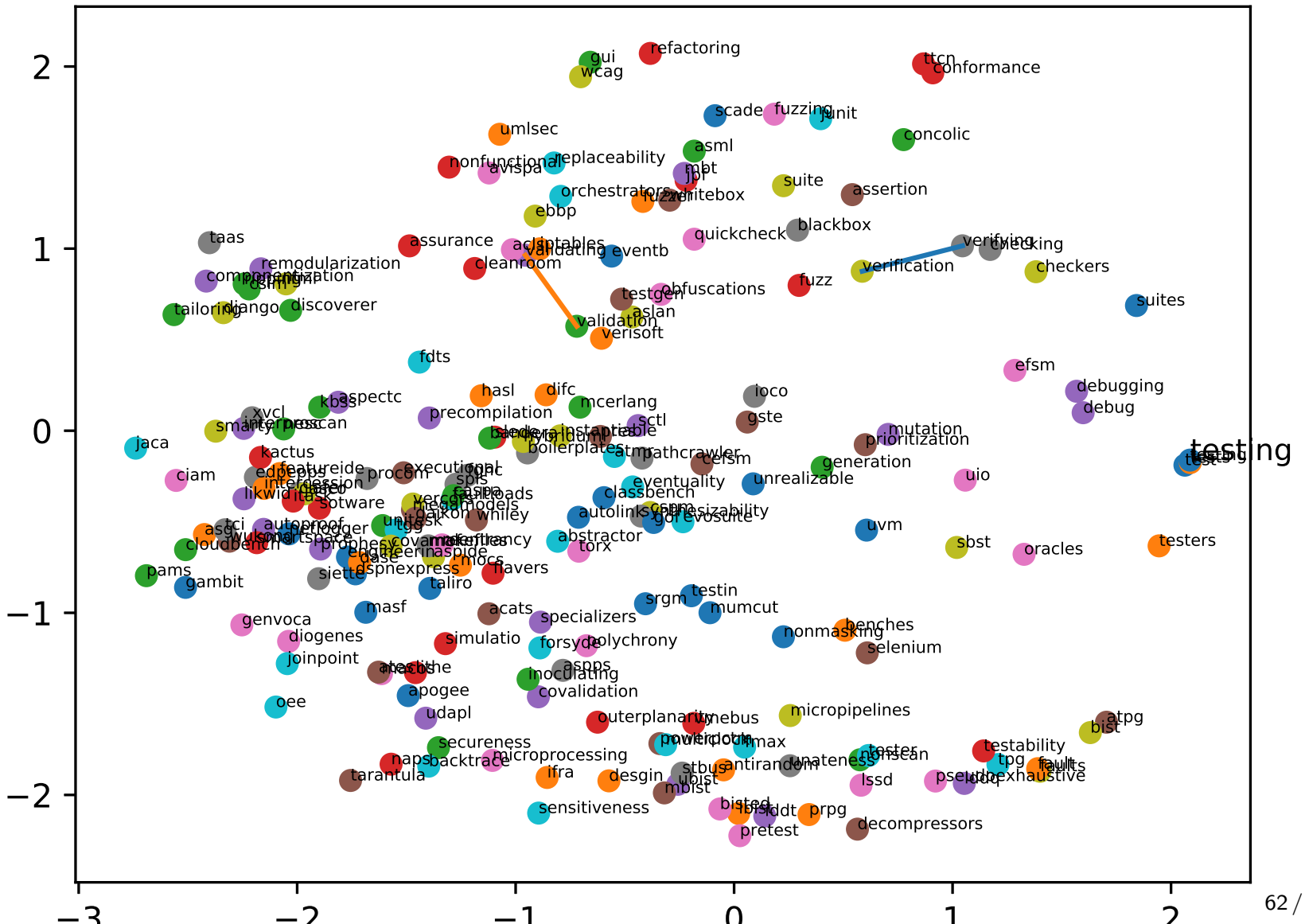
Most similar words to 'testing'

- trained using CBOW. Min-count=20



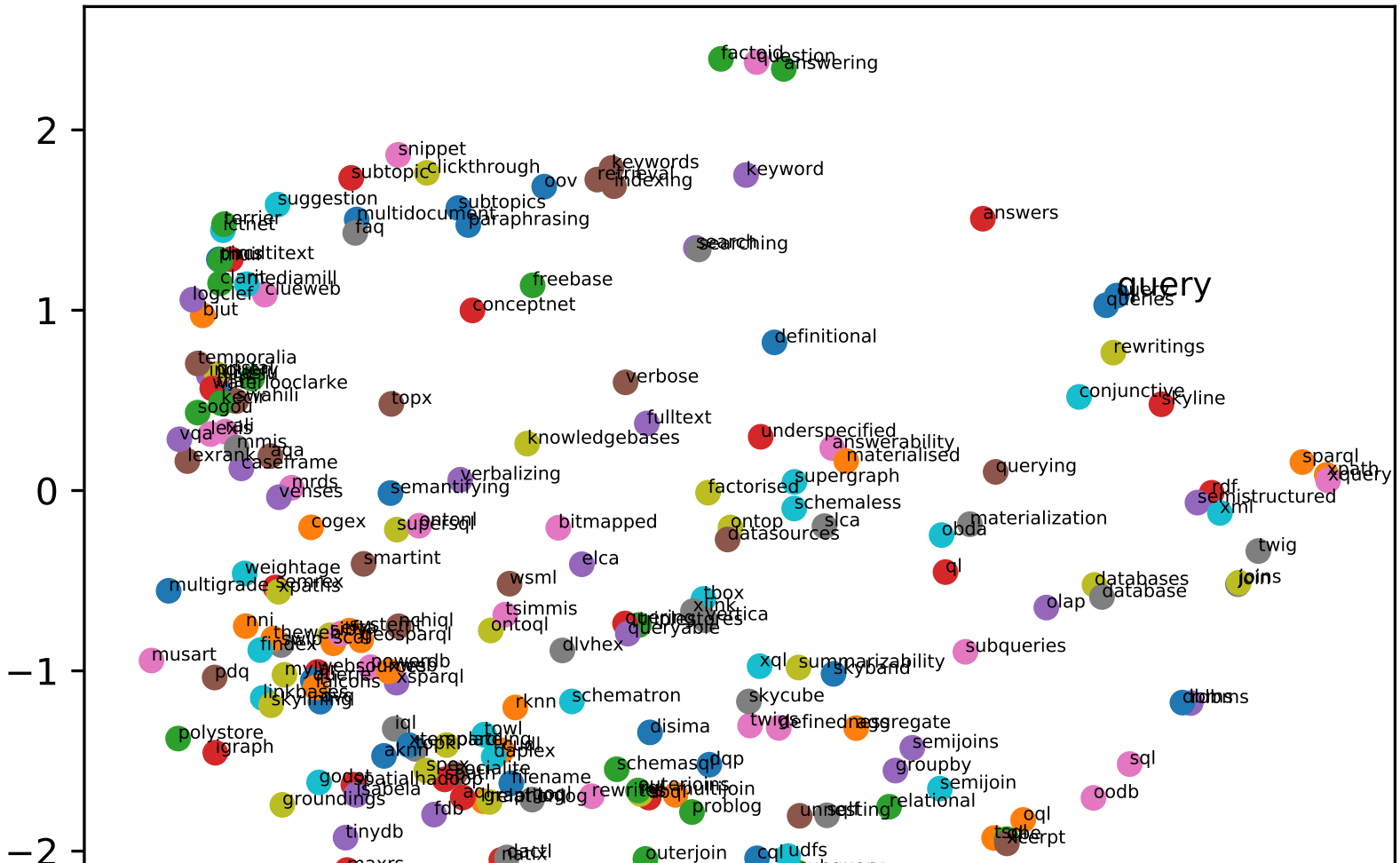
Most similar words to 'testing'

trained using SG. Min-count=5. Analogy is not as good as CBOW.

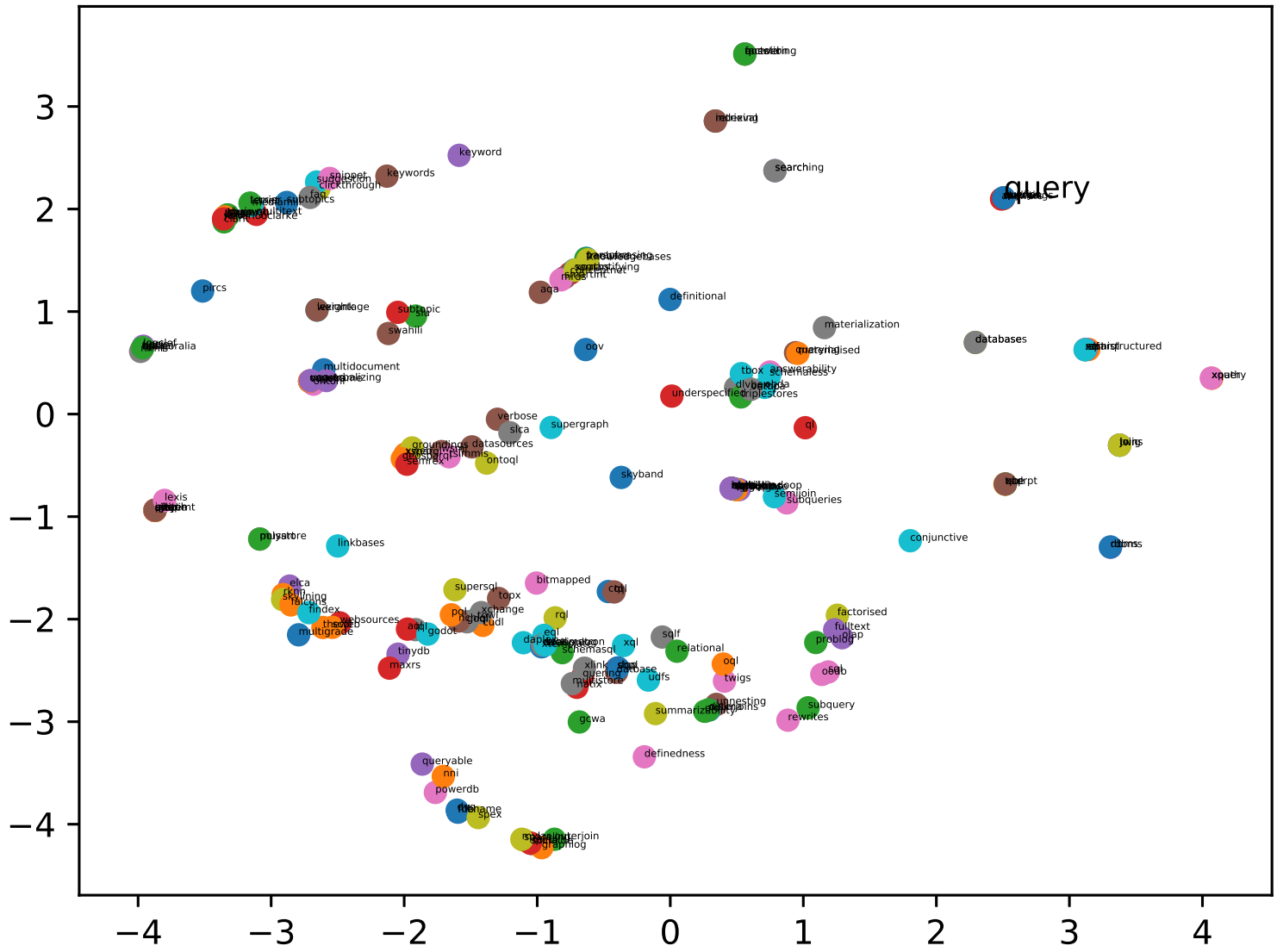


Most similar words to 'queries'

- Same as before. Top 200 most similar words
- Note the close pairs (query, queries), (join, joins), (database, databases)
- there are clusters of research areas, e.g., (rdf, sparql, xpath, xml, semistructured)



Perplexity changes from 1 to 200. Initialized with PCA.



Generate animation

- Generate animated gif
- Latex can also generate animated pdf
- Use animation to show the impact of hyper parameters in t-SNE, word2vec, doc2vec, node2vec
  - epochs, learning rate, min-count, subsampling, ...

```
\usepackage{animate}
...
\animategraphics[autoplay, loop, width=4.5in]{12}{w2v_dblp_sg}{1}{49}
```

For 10,000 ICSE paper titles

```
>>> model = gensim.models.Word2Vec(sentenceList, sg=1, min_count=2, iter=5)
>>> print str(model.most_similar('testing'))
[('real-time', 0.9850457310676575),
('specification', 0.9829429388046265),
('integration', 0.9796831607818604),
('verification', 0.9785716533660889), ('driven', 0.9782571792602539),
('methodology', 0.9776937961578369), ('uml', 0.9774887561798096),
('supporting', 0.9769802093505859), ('building', 0.9768860936164856),
('object-oriented', 0.976362943649292)]
```

More iterations:

```
print str(model.most_similar('testing'))
[('regression', 0.4438987374305725),
('whitebox', 0.4174774650642395),
('test', 0.411671221256261),
 ('c/c++', 0.404751181602478),
 ('model-based', 0.401504337787628 2),
 ('understand', 0.4012585878372 1924),
 ('prioritization', 0.398231685161 5906),
 ('mutation', 0.3904681503772 7356),
 ('verification', 0.3879749178886 4136),
 ('feedback-directed', 0.3781320452690 1245)]
```

Run in parallel

```
model = Word2Vec(sentences, workers=4)
```

The workers parameter has only effect if you have Cython installed.

Summary of Parameters of Word2Vec

- size=100,
- alpha=0.025,
- `min_alpha`=0.0001,
- window=5,
- `min_count`=5,
- `max_vocab_size`=None,
- sample=0.001,
- seed=1,
- iter=5,
- workers=3,
- sg=0,
- negative=5,

Parameters-negative sampling

In Negative sampling, the objective function is:

$$\log \sigma(\mathbf{v}_{w_t} \cdot \mathbf{v}'_{w_c}) + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_n(w)} \left[\log \sigma(-\mathbf{v}_{w_t} \cdot \mathbf{v}'_{w_k}) \right] \tag{7}$$

- negative=5 means that k=5 in the above equation.
- $P_n(w)$ is a parameter.
- Simple case: $P_n(w)$ is the unigram distribution U(w)
- Better to raise to the 3/4rd power (i.e., $U(w)^{3/4}$

Softmax vs. Negative Sampling

$$J_{\text{softmax}} = \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log P(w_{t+j} \mid w_t) \tag{8}$$

$$P(w_i \mid w_t) = \frac{\exp(\mathbf{v}_{w_t} \cdot \mathbf{v}'_{w_i})}{\sum_{j=1}^{V} \exp(\mathbf{v}_{w_t} \cdot \mathbf{v}'_{w_j})} \tag{9}$$

$$\log \sigma(\mathbf{v}_{w_t} \cdot \mathbf{v}'_{w_c}) + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_n(w)} \left[\log \sigma(-\mathbf{v}_{w_t} \cdot \mathbf{v}'_{w_k}) \right] \tag{10}$$

Why epoch is normally above 5?

It is related with window size.

...	government	debt	problems	turning	into	**banking**	crises	as	has	happened

Memory

- word2vec model parameters are stored as matrices (NumPy arrays).
- Each array is #vocabulary (controlled by *mincount* parameter) times #size (size parameter) of floats (single precision aka 4 bytes).

$$10^5 \ words \times 2 \times 100 \ dimensions \ \times 4 \times 3 \approx 229 MB \tag{11}$$

Each word need two vectors.

Save the model for later use

- large corpus may take long time to train a model.
  - dblp_title.txt takes minutes on Macbook Pro.
  - MAS titles take a few hours on a big server.
  - Time depends on hyper-parameters, e.g., epochs, min-count, learning rate, subsampling etc.
- better train once
- also good for others to replicate your experiment

```
model.save('/tmp/mymodel')
new_model = gensim.models.Word2Vec.load('/tmp/mymodel')
```

Test on other data sources (e.g.NLTK)

```
>>> from gensim.models import Word2Vec
>>> from nltk.corpus import brown, movie_reviews, treebank
>>> b = Word2Vec(brown.sents())
>>> mr = Word2Vec(movie_reviews.sents())
>>> t = Word2Vec(treebank.sents())

>>> b.most_similar('money', topn=5)
[('pay', 0.6832243204116821), ('ready', 0.6152011156082153), ('try', 0.584539294
```

```
>>> model.wv.doesnt_match(['testing', 'verification', 'software', 'windsor', 'en

'windsor'
```

Evaluation using standard test sets

```
model.accuracy('/tmp/questions-words.txt')
2014-02-01 22:14:28,387 : INFO : family: 88.9% (304/342)
2014-02-01 22:29:24,006 : INFO : gram1-adjective-to-adverb: 32.4% (263/812)
2014-02-01 22:36:26,528 : INFO : gram2-opposite: 50.3% (191/380)
2014-02-01 23:00:52,406 : INFO : gram3-comparative: 91.7% (1222/1332)
2014-02-01 23:13:48,243 : INFO : gram4-superlative: 87.9% (617/702)
2014-02-01 23:29:52,268 : INFO : gram5-present-participle: 79.4% (691/870)
2014-02-01 23:57:04,965 : INFO : gram7-past-tense: 67.1% (995/1482)
2014-02-02 00:15:18,525 : INFO : gram8-plural: 89.6% (889/992)
2014-02-02 00:28:18,140 : INFO : gram9-plural-verbs: 68.7% (482/702)
2014-02-02 00:28:18,140 : INFO : total: 74.3% (5654/7614)
```

References I

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, pages 855–864. ACM, 2016.

Soumyajit Ganguly and Vikram Pudi. Paper2vec: Combining graph and text information for scientific paper representation. 2017.

Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, volume 14, pages 1188–1196, 2014.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

References II

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *SIGKDD*, pages 701–710. ACM, 2014.

Jian Tang, Meng Qu, and Qiaozhu Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *SIGKDD*, pages 1165–1174. ACM, 2015.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, pages 1067–1077. ACM, 2015.

Suhang Wang, Jiliang Tang, Charu Aggarwal, and Huan Liu. Linked document embedding for classification. In *CIKM*, pages 115–124. ACM, 2016.

Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y Chang. Network representation learning with rich text information. In *IJCAI*, pages 2111–2117, 2015.

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As there wa
some unprocessed data that should have been added to the final page this
extra page has been added to receive it.
If you rerun the document (without altering it) this surplus page will go aw
because LaTeX now knows how many pages to expect for this document.