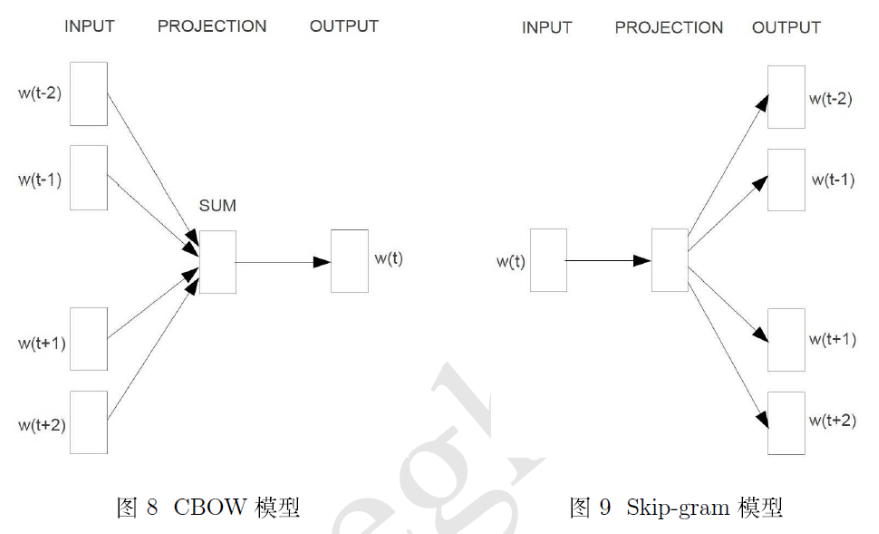
<http://radimrehurek.com/gensim/models/word2vec.html>

gensim API指南：

models.word2vec—使用word2vec的深度学习技术

深入学习的Word2vec “skip-gram 和CBOW模型”，采用分层softmax或负采样。



最初的训练算法从C程序包<https://code.google.com/p/word2vec/> 移植而来的，并扩展了功能。

关于gensim word2vec的博客教程，交互式的应用谷歌新闻进行训练的Web应用程序http://radimrehurek.com/2014/02/word2vec-tutorial/ Googlenews

确保在安装GENSIM之前有C编译器，使用优化（编译）Word2vec训练（70倍的速度相比于原先单纯用NumPy实现的[ 3 ]）

例如初始化一个模型：

**>>>** model = Word2Vec(sentences, size=100, window=5, min\_count=5, workers=4)

保存模型到磁盘：

**>>>** model.save(fname)

**>>>** model = Word2Vec.load(fname) *# you can continue training with the loaded model!*

模型还可以从磁盘文件以Word2vec C的格式实例化：

**>>>** model = Word2Vec.load\_word2vec\_format('/tmp/vectors.txt', binary=False) *# C text format*

**>>>** model = Word2Vec.load\_word2vec\_format('/tmp/vectors.bin', binary=True) *# C binary format*

可以用该模型进行NLP各种句法/语义自然语言处理任务。内置了其中部分：

**>>>** model.most\_similar(positive=['woman', 'king'], negative=['man'])

[('queen', 0.50882536), ...]

**>>>** model.doesnt\_match("breakfast cereal dinner lunch".split())

'cereal'

**>>>** model.similarity('woman', 'man')

0.73723527

**>>>** model['computer'] *# raw numpy vector of a word*

array([-0.00449447, -0.00310097, 0.02421786, ...], dtype=float32)

如果已经训练了一个模型（=不再进行更新，只有查询操作），可以：

**>>>** model.init\_sims(replace=True)

削减模型中不必要的内存=使用（多）更少的RAM。

有个gensim.models.phrases模块，可以自动检测短语超过一个字。使用这个，可以发现Word2vec模型，“words”实际上是多词表达，如new\_york\_times或financial\_crisis：

**>>>** bigram\_transformer = gensim.models.Phrases(sentences)

**>>>** model = Word2Vec(bigram\_transformer[sentences], size=100, ...)

|  |  |
| --- | --- |
| [**[1]**](http://radimrehurek.com/gensim/models/word2vec.html#id1) | Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In Proceedings of Workshop at ICLR, 2013. |

|  |  |
| --- | --- |
| [**[2]**](http://radimrehurek.com/gensim/models/word2vec.html#id2) | Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NIPS, 2013. |

|  |  |
| --- | --- |
| [**[3]**](http://radimrehurek.com/gensim/models/word2vec.html#id3) | Optimizing word2vec in gensim, [**http://radimrehurek.com/2013/09/word2vec-in-python-part-two-optimizing/**](http://radimrehurek.com/2013/09/word2vec-in-python-part-two-optimizing/) |

*class*gensim.models.word2vec.BrownCorpus(*dirname*)

Bases: object

Iterate over sentences from the Brown corpus (part of NLTK data).

*class*gensim.models.word2vec.LineSentence(*source*, *max\_sentence\_length=10000*, *limit=None*)

Bases: object

Simple format: one sentence = one line; words already preprocessed and separated by whitespace.

*source* can be either a string or a file object. Clip the file to the first *limit* lines (or no clipped if limit is None, the default).

Example:

sentences = LineSentence('myfile.txt')

Or for compressed files:

sentences = LineSentence('compressed\_text.txt.bz2')

sentences = LineSentence('compressed\_text.txt.gz')

classgensim.models.word2vec.Text8Corpus(fname, max\_sentence\_length=10000)

Bases: object

Iterate over sentences from the “text8” corpus, unzipped from [**http://mattmahoney.net/dc/text8.zip**](http://mattmahoney.net/dc/text8.zip) .

classgensim.models.word2vec.Vocab(\*\*kwargs)

Bases: object

A single vocabulary item, used internally for collecting per-word frequency/sampling info, and for constructing binary trees (incl. both word leaves and inner nodes).

classgensim.models.word2vec.Word2Vec(sentences=None, size=100, alpha=0.025, window=5, min\_count=5, max\_vocab\_size=None,sample=0.001, seed=1, workers=3, min\_alpha=0.0001, sg=0, hs=0, negative=5, cbow\_mean=1, hashfxn=<built-in function hash>, iter=5,null\_word=0, trim\_rule=None, sorted\_vocab=1, batch\_words=10000)

Bases: [**gensim.utils.SaveLoad**](http://radimrehurek.com/gensim/utils.html#gensim.utils.SaveLoad)

Class for training, using and evaluating neural networks described in [**https://code.google.com/p/word2vec/**](https://code.google.com/p/word2vec/)

The model can be stored/loaded via its save() and load() methods, or stored/loaded in a format compatible with the original word2vec implementation via save\_word2vec\_format() and load\_word2vec\_format().

Initialize the model from an iterable of sentences. Each sentence is a list of words (unicode strings) that will be used for training.

The sentences iterable can be simply a list, but for larger corpora, consider an iterable that streams the sentences directly from disk/network. See [**BrownCorpus**](http://radimrehurek.com/gensim/models/word2vec.html#gensim.models.word2vec.BrownCorpus), [**Text8Corpus**](http://radimrehurek.com/gensim/models/word2vec.html#gensim.models.word2vec.Text8Corpus) or [**LineSentence**](http://radimrehurek.com/gensim/models/word2vec.html#gensim.models.word2vec.LineSentence) in this module for such examples.

If you don’t supply sentences, the model is left uninitialized – use if you plan to initialize it in some other way.

sg defines the training algorithm. By default (sg=0), CBOW is used. Otherwise (sg=1), skip-gram is employed.

size is the dimensionality of the feature vectors.

window is the maximum distance between the current and predicted word within a sentence.

alpha is the initial learning rate (will linearly drop to min\_alpha as training progresses).

seed = for the random number generator. Initial vectors for each word are seeded with a hash of the concatenation of word + str(seed). Note that for a fully deterministically-reproducible run, you must also limit the model to a single worker thread, to eliminate ordering jitter from OS thread scheduling. (In Python 3, reproducibility between interpreter launches also requires use of the PYTHONHASHSEED environment variable to control hash randomization.)

min\_count = ignore all words with total frequency lower than this.

max\_vocab\_size = limit RAM during vocabulary building; if there are more unique words than this, then prune the infrequent ones. Every 10 million word types need about 1GB of RAM. Set to None for no limit (default).

sample = threshold for configuring which higher-frequency words are randomly downsampled;

default is 1e-3, useful range is (0, 1e-5).

workers = use this many worker threads to train the model (=faster training with multicore machines).

hs = if 1, hierarchical softmax will be used for model training. If set to 0 (default), and negative is non-zero, negative sampling will be used.

negative = if > 0, negative sampling will be used, the int for negative specifies how many “noise words” should be drawn (usually between 5-20). Default is 5. If set to 0, no negative samping is used.

cbow\_mean = if 0, use the sum of the context word vectors. If 1 (default), use the mean. Only applies when cbow is used.

hashfxn = hash function to use to randomly initialize weights, for increased training reproducibility. Default is Python’s rudimentary built in hash function.

iter = number of iterations (epochs) over the corpus. Default is 5.

trim\_rule = vocabulary trimming rule, specifies whether certain words should remain in the vocabulary, be trimmed away, or handled using the default (discard if word count < min\_count). Can be None (min\_count will be used), or a callable that accepts parameters (word, count, min\_count) and returns either utils.RULE\_DISCARD, utils.RULE\_KEEP or utils.RULE\_DEFAULT. Note: The rule, if given, is only used prune vocabulary during build\_vocab() and is not stored as part of the model.

sorted\_vocab = if 1 (default), sort the vocabulary by descending frequency before assigning word indexes.

batch\_words = target size (in words) for batches of examples passed to worker threads (and thus cython routines). Default is 10000. (Larger batches will be passed if individual texts are longer than 10000 words, but the standard cython code truncates to that maximum.)

accuracy(questions, restrict\_vocab=30000, most\_similar=<function most\_similar>, case\_insensitive=True)

Compute accuracy of the model. questions is a filename where lines are 4-tuples of words, split into sections by ”: SECTION NAME” lines. See questions-words.txt in [**https://storage.googleapis.com/google-code-archive-source/v2/code.google.com/word2vec/source-archive.zip**](https://storage.googleapis.com/google-code-archive-source/v2/code.google.com/word2vec/source-archive.zip) for an example.

The accuracy is reported (=printed to log and returned as a list) for each section separately, plus there’s one aggregate summary at the end.

Use restrict\_vocab to ignore all questions containing a word not in the first restrict\_vocab words (default 30,000). This may be meaningful if you’ve sorted the vocabulary by descending frequency. In case case\_insensitive is True, the first restrict\_vocab words are taken first, and then case normalization is performed.

Use case\_insensitive to convert all words in questions and vocab to their uppercase form before evaluating the accuracy (default True). Useful in case of case-mismatch between training tokens and question words. In case of multiple case variants of a single word, the vector for the first occurrence (also the most frequent if vocabulary is sorted) is taken.

This method corresponds to the compute-accuracy script of the original C word2vec.

build\_vocab(sentences, keep\_raw\_vocab=False, trim\_rule=None, progress\_per=10000)

Build vocabulary from a sequence of sentences (can be a once-only generator stream). Each sentence must be a list of unicode strings.

clear\_sims()

create\_binary\_tree()

Create a binary Huffman tree using stored vocabulary word counts. Frequent words will have shorter binary codes. Called internally from build\_vocab().

doesnt\_match(words)

Which word from the given list doesn’t go with the others?

Example:

**>>>** trained\_model.doesnt\_match("breakfast cereal dinner lunch".split())

'cereal'

estimate\_memory(vocab\_size=None, report=None)

Estimate required memory for a model using current settings and provided vocabulary size.

finalize\_vocab()

Build tables and model weights based on final vocabulary settings.

init\_sims(replace=False)

Precompute L2-normalized vectors.

If replace is set, forget the original vectors and only keep the normalized ones = saves lots of memory!

Note that you **cannot continue training** after doing a replace. The model becomes effectively read-only = you can callmost\_similar, similarity etc., but not train.

intersect\_word2vec\_format(fname, lockf=0.0, binary=False, encoding='utf8', unicode\_errors='strict')

Merge the input-hidden weight matrix from the original C word2vec-tool format given, where it intersects with the current vocabulary. (No words are added to the existing vocabulary, but intersecting words adopt the file’s weights, and non-intersecting words are left alone.)

binary is a boolean indicating whether the data is in binary word2vec format.

lockf is a lock-factor value to be set for any imported word-vectors; the default value of 0.0 prevents further updating of the vector during subsequent training. Use 1.0 to allow further training updates of merged vectors.

classmethodload(\*args, \*\*kwargs)

classmethodload\_word2vec\_format(fname, fvocab=None, binary=False, encoding='utf8', unicode\_errors='strict', limit=None, datatype=<type 'numpy.float32'>)

Load the input-hidden weight matrix from the original C word2vec-tool format.

Note that the information stored in the file is incomplete (the binary tree is missing), so while you can query for word similarity etc., you cannot continue training with a model loaded this way.

binary is a boolean indicating whether the data is in binary word2vec format. norm\_only is a boolean indicating whether to only store normalised word2vec vectors in memory. Word counts are read from fvocab filename, if set (this is the file generated by -save-vocabflag of the original C tool).

If you trained the C model using non-utf8 encoding for words, specify that encoding in encoding.

unicode\_errors, default ‘strict’, is a string suitable to be passed as the errors argument to the unicode() (Python 2.x) or str() (Python 3.x) function. If your source file may include word tokens truncated in the middle of a multibyte unicode character (as is common from the original word2vec.c tool), ‘ignore’ or ‘replace’ may help.

limit sets a maximum number of word-vectors to read from the file. The default, None, means read all.

datatype (experimental) can coerce dimensions to a non-default float type (such as np.float16) to save memory. (Such types may result in much slower bulk operations or incompatibility with optimized routines.)

staticlog\_accuracy(section)

make\_cum\_table(power=0.75, domain=2147483647)

Create a cumulative-distribution table using stored vocabulary word counts for drawing random words in the negative-sampling training routines.

To draw a word index, choose a random integer up to the maximum value in the table (cum\_table[-1]), then finding that integer’s sorted insertion point (as if by bisect\_left or ndarray.searchsorted()). That insertion point is the drawn index, coming up in proportion equal to the increment at that slot.

Called internally from ‘build\_vocab()’.

most\_similar(positive=[], negative=[], topn=10, restrict\_vocab=None, indexer=None)

Find the top-N most similar words. Positive words contribute positively towards the similarity, negative words negatively.

This method computes cosine similarity between a simple mean of the projection weight vectors of the given words and the vectors for each word in the model. The method corresponds to the word-analogy and distance scripts in the original word2vec implementation.

If topn is False, most\_similar returns the vector of similarity scores.

restrict\_vocab is an optional integer which limits the range of vectors which are searched for most-similar values. For example, restrict\_vocab=10000 would only check the first 10000 word vectors in the vocabulary order. (This may be meaningful if you’ve sorted the vocabulary by descending frequency.)

Example:

**>>>** trained\_model.most\_similar(positive=['woman', 'king'], negative=['man'])

[('queen', 0.50882536), ...]

most\_similar\_cosmul(positive=[], negative=[], topn=10)

Find the top-N most similar words, using the multiplicative combination objective proposed by Omer Levy and Yoav Goldberg in [**[4]**](http://radimrehurek.com/gensim/models/word2vec.html#id8). Positive words still contribute positively towards the similarity, negative words negatively, but with less susceptibility to one large distance dominating the calculation.

In the common analogy-solving case, of two positive and one negative examples, this method is equivalent to the “3CosMul” objective (equation (4)) of Levy and Goldberg.

Additional positive or negative examples contribute to the numerator or denominator, respectively – a potentially sensible but untested extension of the method. (With a single positive example, rankings will be the same as in the default most\_similar.)

Example:

**>>>** trained\_model.most\_similar\_cosmul(positive=['baghdad', 'england'], negative=['london'])

[(u'iraq', 0.8488819003105164), ...]

|  |  |
| --- | --- |
| [**[4]**](http://radimrehurek.com/gensim/models/word2vec.html#id7) | Omer Levy and Yoav Goldberg. Linguistic Regularities in Sparse and Explicit Word Representations, 2014. |

n\_similarity(ws1, ws2)

Compute cosine similarity between two sets of words.

Example:

**>>>** trained\_model.n\_similarity(['sushi', 'shop'], ['japanese', 'restaurant'])

0.61540466561049689

**>>>** trained\_model.n\_similarity(['restaurant', 'japanese'], ['japanese', 'restaurant'])

1.0000000000000004

**>>>** trained\_model.n\_similarity(['sushi'], ['restaurant']) == trained\_model.similarity('sushi', 'restaurant')

True

reset\_from(other\_model)

Borrow shareable pre-built structures (like vocab) from the other\_model. Useful if testing multiple models in parallel on the same corpus.

reset\_weights()

Reset all projection weights to an initial (untrained) state, but keep the existing vocabulary.

save(\*args, \*\*kwargs)

Save the object to file (also see load).

fname\_or\_handle is either a string specifying the file name to save to, or an open file-like object which can be written to. If the object is a file handle, no special array handling will be performed; all attributes will be saved to the same file.

If separately is None, automatically detect large numpy/scipy.sparse arrays in the object being stored, and store them into separate files. This avoids pickle memory errors and allows mmap’ing large arrays back on load efficiently.

You can also set separately manually, in which case it must be a list of attribute names to be stored in separate files. The automatic check is not performed in this case.

ignore is a set of attribute names to not serialize (file handles, caches etc). On subsequent load() these attributes will be set to None.

pickle\_protocol defaults to 2 so the pickled object can be imported in both Python 2 and 3.

save\_word2vec\_format(fname, fvocab=None, binary=False)

Store the input-hidden weight matrix in the same format used by the original C word2vec-tool, for compatibility.

fname is the file used to save the vectors in fvocab is an optional file used to save the vocabulary binary is an optional boolean indicating whether the data is to be saved in binary word2vec format (default: False)

scale\_vocab(min\_count=None, sample=None, dry\_run=False, keep\_raw\_vocab=False, trim\_rule=None)

Apply vocabulary settings for min\_count (discarding less-frequent words) and sample (controlling the downsampling of more-frequent words).

Calling with dry\_run=True will only simulate the provided settings and report the size of the retained vocabulary, effective corpus length, and estimated memory requirements. Results are both printed via logging and returned as a dict.

Delete the raw vocabulary after the scaling is done to free up RAM, unless keep\_raw\_vocab is set.

scan\_vocab(sentences, progress\_per=10000, trim\_rule=None)

Do an initial scan of all words appearing in sentences.

score(sentences, total\_sentences=1000000, chunksize=100, queue\_factor=2, report\_delay=1)

Score the log probability for a sequence of sentences (can be a once-only generator stream). Each sentence must be a list of unicode strings. This does not change the fitted model in any way (see Word2Vec.train() for that).

We have currently only implemented score for the hierarchical softmax scheme, so you need to have run word2vec with hs=1 and negative=0 for this to work.

Note that you should specify total\_sentences; we’ll run into problems if you ask to score more than this number of sentences but it is inefficient to set the value too high.

See the article by [**[taddy]**](http://radimrehurek.com/gensim/models/word2vec.html#taddy) and the gensim demo at [**[deepir]**](http://radimrehurek.com/gensim/models/word2vec.html#deepir) for examples of how to use such scores in document classification.

|  |  |
| --- | --- |
| [**[taddy]**](http://radimrehurek.com/gensim/models/word2vec.html#id9) | Taddy, Matt. Document Classification by Inversion of Distributed Language Representations, in Proceedings of the 2015 Conference of the Association of Computational Linguistics. |

|  |  |
| --- | --- |
| [**[deepir]**](http://radimrehurek.com/gensim/models/word2vec.html#id10) | [**https://github.com/piskvorky/gensim/blob/develop/docs/notebooks/deepir.ipynb**](https://github.com/piskvorky/gensim/blob/develop/docs/notebooks/deepir.ipynb) |

seeded\_vector(seed\_string)

Create one ‘random’ vector (but deterministic by seed\_string)

similar\_by\_vector(vector, topn=10, restrict\_vocab=None)

Find the top-N most similar words by vector.

If topn is False, similar\_by\_vector returns the vector of similarity scores.

restrict\_vocab is an optional integer which limits the range of vectors which are searched for most-similar values. For example, restrict\_vocab=10000 would only check the first 10000 word vectors in the vocabulary order. (This may be meaningful if you’ve sorted the vocabulary by descending frequency.)

Example:

**>>>** trained\_model.similar\_by\_vector([1,2])

[('survey', 0.9942699074745178), ...]

similar\_by\_word(word, topn=10, restrict\_vocab=None)

Find the top-N most similar words.

If topn is False, similar\_by\_word returns the vector of similarity scores.

restrict\_vocab is an optional integer which limits the range of vectors which are searched for most-similar values. For example, restrict\_vocab=10000 would only check the first 10000 word vectors in the vocabulary order. (This may be meaningful if you’ve sorted the vocabulary by descending frequency.)

Example:

**>>>** trained\_model.similar\_by\_word('graph')

[('user', 0.9999163150787354), ...]

similarity(w1, w2)

Compute cosine similarity between two words.

Example:

**>>>** trained\_model.similarity('woman', 'man')

0.73723527

**>>>** trained\_model.similarity('woman', 'woman')

1.0

sort\_vocab()

Sort the vocabulary so the most frequent words have the lowest indexes.

train(sentences, total\_words=None, word\_count=0, total\_examples=None, queue\_factor=2, report\_delay=1.0)

Update the model’s neural weights from a sequence of sentences (can be a once-only generator stream). For Word2Vec, each sentence must be a list of unicode strings. (Subclasses may accept other examples.)

To support linear learning-rate decay from (initial) alpha to min\_alpha, either total\_examples (count of sentences) or total\_words (count of raw words in sentences) should be provided, unless the sentences are the same as those that were used to initially build the vocabulary.

wmdistance(document1, document2)

Compute the Word Mover’s Distance between two documents. When using this code, please consider citing the following papers:

Note that if one of the documents have no words that exist in the Word2Vec vocab, float(‘inf’) (i.e. infinity) will be returned.

This method only works if pyemd is installed (can be installed via pip, but requires a C compiler).

Example:

**>>>** *# Train word2vec model.*

**>>>** model = Word2Vec(sentences)

**>>>** *# Some sentences to test.*

**>>>** sentence\_obama = 'Obama speaks to the media in Illinois'.lower().split()

**>>>** sentence\_president = 'The president greets the press in Chicago'.lower().split()

**>>>** *# Remove their stopwords.*

**>>> from** **nltk.corpus** **import** stopwords

**>>>** stopwords = nltk.corpus.stopwords.words('english')

**>>>** sentence\_obama = [w **for** w **in** sentence\_obama **if** w **not** **in** stopwords]

**>>>** sentence\_president = [w **for** w **in** sentence\_president **if** w **not** **in** stopwords]

**>>>** *# Compute WMD.*

**>>>** distance = model.wmdistance(sentence\_obama, sentence\_president)

gensim.models.word2vec.score\_cbow\_pair(model, word, word2\_indices, l1)

gensim.models.word2vec.score\_sg\_pair(model, word, word2)

gensim.models.word2vec.train\_cbow\_pair(model, word, input\_word\_indices, l1, alpha, learn\_vectors=True, learn\_hidden=True)

gensim.models.word2vec.train\_sg\_pair(model, word, context\_index, alpha, learn\_vectors=True, learn\_hidden=True, context\_vectors=None,context\_locks=None)