

# Speed Up Gensim's Word2vec for a Massive Dataset

Asked 1 year, 11 months ago   Active 1 year, 11 months ago   Viewed 3k times

3 I'm trying to build a Word2vec (or FastText) model using Gensim on a massive dataset which is composed of 1000 files, each contains ~210,000 sentences, and each sentence contains ~1000 words. The training was made on a 185gb RAM, 36-core machine. I validated that

```
gensim.models.word2vec.FAST_VERSION == 1
```

1 First, I've tried the following:

```
files = gensim.models.word2vec.PathLineSentences('path/to/files')
model = gensim.models.word2vec.Word2Vec(files, workers=-1)
```

But after 13 hours I decided it is running for too long and stopped it.

Then I tried building the vocabulary based on a single file, and train based on all 1000 files as follows:

```
files = os.listdir('path/to/files')
model = gensim.models.word2vec.Word2Vec(min_count=1, workers=-1)
model.build_vocab(corpus_file=files[0])
for file in files:
    model.train(corpus_file=file, total_words=model.corpus_total_words, epochs=1)
```

But I checked a sample of word vectors before and after the training, and there was no change, which means no actual training was done.

I can use some advise on how to run it quickly and successfully. Thanks!

## Update #1:

Here is the code to check vector updates:

```
file = 'path/to/single/gzipped/file'
total_words = 197264406 # number of words in 'file'
total_examples = 209718 # number of records in 'file'
model = gensim.models.word2vec.Word2Vec(iter=5, workers=12)
model.build_vocab(corpus_file=file)
wv_before = model.wv['9995']
model.train(corpus_file=file, total_words=total_words, total_examples=total_examples, epochs=5)
wv_after = model.wv['9995']
```

so the vectors: `wv_before` and `wv_after` are exactly the same

gensim word2vec fasttext

Share Follow

edited Jan 28 '20 at 12:18

asked Jan 26 '20 at 14:51



Kamaney

75 ● 2 ● 7

1 Answer

Active Oldest Votes

There's no facility in `gensim`'s `Word2Vec` to accept a **negative** value for `workers`. (Where'd you get the idea

that would be meaningful?)

So, it's quite possible that's breaking something, perhaps preventing any training from even being attempted.



Was there sensible logging output (at level `INFO`) suggesting that training was progressing in your trial runs, either against the `PathLineSentences` or your second attempt? Did utilities like `top` show busy threads? Did the output suggest a particular rate of progress & let you project-out a likely finishing time?

I'd suggest using a positive `workers` value and watching `INFO`-level logging to get a better idea what's happening.

Unfortunately, even with 36 cores, using a corpus iterable sequence (like `PathLineSentences`) puts gensim `Word2Vec` in a model where you'll likely get maximum throughput with a `workers` value in the 8-16 range, using far less than all your threads. But it will do the right thing, on a corpus of any size, even if it's being assembled by the iterable sequence on-the-fly.

Using the `corpus_file` mode can saturate far more cores, but you should still specify the actual number of worker threads to use – in your case, `workers=36` – and it is designed to work on from a single file with all data.

Your code which attempts to `train()` many times with `corpus_file` has lots of problems, and I can't think of a way to adapt `corpus_file` mode to work on your many files. Some of the problems include:

- you're only building the vocabulary from the 1st file, which means any words only appearing in other files will be unknown and ignored, and any of the word-frequency-driven parts of the `Word2Vec` algorithm may be working on unrepresentative
- the model builds its estimate of the expected corpus size (eg: `model.corpus_total_words`) from the `build_vocab()` step, so every `train()` will behave as if that size is the total corpus size, in its progress-reporting & management of the internal `alpha` learning-rate decay. So those logs will be wrong, the `alpha` will be mismanaged in a fresh decay each `train()`, resulting in a nonsensical jigsaw up-and-down `alpha` over all files.
- you're only iterating over each file's contents once, which isn't typical. (It might be reasonable in a giant 210-billion word corpus, though, if every file's text is equally and randomly representative of the domain. In that case, the full corpus once might be as good as iterating over a corpus that's 1/5th the size 5 times. But it'd be a problem if some words/patterns-of-usage are all clumped in certain files – the best training interleaves contrasting examples throughout each epoch and all epochs.)
- `min_count=1` is almost always unwise with this algorithm, and especially so in large corpora of typical natural-language word frequencies. Rare words, and especially those appearing only once or a few times, make the model gigantic but those words won't get good word-vectors, and keeping them in acts like noise interfering with the improvement of other more-common words.

I recommend:

Try the corpus iterable sequence mode, with logging and a sensible `workers` value, to at least get an accurate read of how long it might take. (The longest step will be the initial vocabulary scan, which is essentially single-threaded and must visit all data. But you can `.save()` the model after that step, to then later re-`.load()` it, tinker with settings, and try different `train()` approaches without repeating the slow vocabulary survey.)

Try aggressively-higher values of `min_count` (discarding more rare words for a smaller model & faster training). Perhaps also try aggressively-smaller values of `sample` (like `1e-05`, `1e-06`, etc) to discard a larger fraction of the most-frequent words, for faster training that also often improves overall word-vector quality (by spending relatively more effort on less-frequent words).

If it's still too slow, consider if you could using a smaller subsample of your corpus might be enough.

Consider the `corpus_file` method if you can roll much or all of your data into the single file it requires.

Share Follow

answered Jan 27 '20 at 6:50



gojomo

45.1k ●12 ●79 ●102

---

Thanks for the detailed response. I tested your recommendations on a single file before implementing it on the uber file: setting `workers=8`, `epochs=5`, `min_count=5`, and log-level to `INFO`. The progress log showed that a training is happening, and the `top` also showed the busy threads. The word vectors still remain the same before and after training... – [Kamaney](#) Jan 27 '20 at 14:52

---

Per above, if you're using a single file (and `corpus_file`), you might as well use `workers=36`. How are you checking that the word-vectors remain the same? (Watch out for using anything like a `most_similar()` check, as that can cause a cached unit-length-normalized set of vectors to be used, which won't necessarily change after more training.) – [gojomo](#) Jan 27 '20 at 19:32

---

I check a sample of words right after calling instantiating the model and running `model.build_vocab`, and then again after running `model.train`. – [Kamaney](#) Jan 28 '20 at 8:27

---

If the logging shows time-consuming training happening, then the words should be changing – so perhaps there's something wrong with how you're checking the sample of word-vectors. If you show (in your answer) the code you're using to check, the problem might become clear. – [gojomo](#) Jan 28 '20 at 10:11

---

A simple access like `wv_before = model.wv['9995']`, into the underlying `numpy` array, will typically create a new 'view' onto just that row, not an independent copy. So at the end, yes, `wv_before == wv.after` will be `True` – but that's because `wv_before` has been keeping-up with changes. If you **notice** your sample item before/after, you should see a change – or you could ensure a separate copy with `wv_before = model.wv['9995'].copy()`. – [gojomo](#) Jan 28 '20 at 20:04

---