





doc2vec: measurement of performance and 'workers' parameter

Asked 3 years, 1 month ago Active 1 year, 4 months ago Viewed 976 times

- 
3

- I have an awfully large corpora as input to my doc2vec training, around 23mil documents streamed using an iterable function. I was wondering if it were at all possible to see the development of my training progress, possibly through finding out which iteration its currently on, words per second or some similar metric.
- 
1

- I was also wondering how to speed up the performance of doc2vec, other than reducing the size of the corpus. I discovered the `workers` parameter and I'm currently training on 4 processes; the intuition behind this number was that multiprocessing cannot take advantage of virtual cores. I was wondering if this was the case for the doc2vec `workers` parameter or if I could use 8 workers instead or even potentially higher (I have a quad-core processor, running Ubuntu).

I have to add that using the unix command `top -H` reports only around a 15% CPU usage per python process using 8 workers and around 27% CPU usage per process on 4 workers.

`python` `nlp` `multiprocessing` `word2vec` `doc2vec`

Share Follow

asked Dec 5 '18 at 19:13






Alexander Hepburn


618 ●1 ●6 ●23

1 Answer

Active Oldest Votes

- 
4


- If you enable logging at the INFO level you should see copious progress output. Following [gensim's Doc2Vec tutorial](#), that'd look like

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

- 
- The optimal throughput for gensim's `Word2Vec` or `Doc2Vec` models is often at some level of `workers` between 3 and 12, but never more than the number of processor cores available. (There's a further optimization that's especially helpful for machines with many more cores, if you use a specific on-disk corpus format, that's available in the most recent 3.6.0 gensim release – see the [release notes](#) for more info.)

If you're seeing such low utilization on a 4-core, 4-worker setup, the bottleneck might be your [corpus iterator](#). If it's doing any complicated IO or regex-based text-processing, then often the training worker threads are idle waiting for the one master corpus-iterator thread to produce more text, limiting overall utilization & efficiency.

You should try to do the complicated stuff once, and re-write the tagged/tokenized results to disk as a more simple file. Then read that with a very simple line-and-space-delimited iterator for the actual model training.

(If your 4 cores actually support more virtual cores, it's possible that some `workers` value up to 8 might achieve higher throughput... but only trial-and-error, with your specific model parameters, can currently find your local optimum. The optimal value can vary with other parameters like `size`, `window`, `negative`, etc.)

Share Follow

edited Aug 19 '20 at 22:33

answered Dec 7 '18 at 1:20



gojomo

45.1k ● 12 ● 79 ● 102
