Stanford CoreNLP and Memory Issues

Stanford CoreNLP (Java version)

In the Stanford CoreNLP GitHub website we read (https://stanfordnlp.github.io/CoreNLP/memory-time.html):

“**People not infrequently complain that Stanford CoreNLP is slow or takes a ton of memory.** In some configurations this is true. In other configurations, this is not true….”

Where does all the memory go when using Stanford CoreNLP?

There are three big places that memory goes:

1. Large machine learning models (mainly arrays and maps of Strings for features and floats or doubles for parameters) which are stored in memory (**annotator type matters!**)
2. The annotated document that is stored in memory (**document size matters!**).
3. Very large data structures in memory that are used by some NLP algorithms (**sentence size matters!**)

Secret number 1 for memory problems: The annotator you choose matters

“How slow and memory intensive CoreNLP is depends on the annotators you choose. This is the first rule. … Of course, sometimes the choices that are fast and memory efficient aren’t the choices that produce the highest quality annotations. Sometimes you have to make trade-offs.”

“Some uses of CoreNLP don’t need much time or space. It can just tokenize and sentence split text using very little time and space. It can do this on the sample text while giving Java just 20MB of memory…. So, the first thing to know is that **CoreNLP will be slow and take a lot of memory if and only if you choose annotators and annotation options that are slow and use a lot of memory**.”

Currently, the largest models in the default pipeline are the neural networks for statistical coreference.

The shift-reduce constituency parser also has very large models. If you run without them, you can annotate the sample document in 2GB of RAM. … But once you include coreference … then the system really needs 4GB of RAM for this document (and if you run constituency parsing and coreference on a large document, you can easily need 5–6GB of RAM).

So… if a parser blows up your memory, a better way to lessen memory use is to use a different parser.

The Java memory grabber

Java is the main culprit in gobbling up memory. “Strings are internally memory-expensive in Java. Each token is represented as an Object, which stores various token attributes, such as token offsets, which are themselves represented as Objects. **These Java tasks use up plenty of memory.**

Secret number 2 for memory problems: Limit document size

“A whole “document” is represented in memory while processing it. Therefore, if you have a large file, like a novel, the next secret to reducing memory usage is to not treat the whole file as a “document”. **Process a large file a piece, say a chapter, at a time, not all at once.”**

Secret number 3 for memory problems: Limit sentence size

The Stanford CoreNLP traditional englishPCFG.ser.gz constituency parsing takes memory space proportional to the square of the longest sentence length, with a large constant factor. “Parsing sentences that are hundreds of words long will take additional gigabytes of memory just for the parser data structures. **The easiest fix for that is just to not parse super-long sentences.** You can do that with a property like: **-parse.maxlen** 70. This can be a fine solution for something like web pages or newswire, where anything over 70 words is likely a table or list or something that isn’t a real sentence. However, it is unappealing for James Joyce: Several of the sentences in Chapter 13 are over 100 words but are well-formed, proper sentences. For example, here is one of the longer sentences in the chapter:

> Her griddlecakes done to a goldenbrown hue and queen Ann’s pudding of delightful creaminess had won golden opinions from all because she had a lucky hand also for lighting a fire, dredge in the fine selfraising flour and always stir in the same direction,

> then cream the milk and sugar and whisk well the white of eggs though she didn't like the eating part when there were any people that made her shy and often she wondered why you couldn't eat something poetical like violets or roses and they would have a beautifully appointed drawingroom with pictures and engravings and the photograph of grandpapa Giltrap's lovely dog Garryowen that almost talked it was so human and chintz covers for the chairs and that silver toastrack in Clery's summer jumble sales like they have in rich houses.

Nevertheless, in general, very long sentences blow out processing time and memory. One thing to be aware of is that CoreNLP currently uses simple, heuristic sentence splitting on sentence terminators like '.' and '?'. If you are parsing "noisy" text without explicit sentence breaks – this often happens if you parse things like tables or web pages – you can end up with "sentences" more than 500 words long, which it isn't even useful to try to parse. You should either clean these up in data preprocessing or limit the sentence length that annotators try to process. Several annotators support a maximum sentence length property and will simply skip processing of longer sentence. The most commonly useful of these is **parse.maxlen** but there is also **kbp.maxlen**, **ner.maxlen**, and **pos.maxlen**."

Find out how much memory your computer has

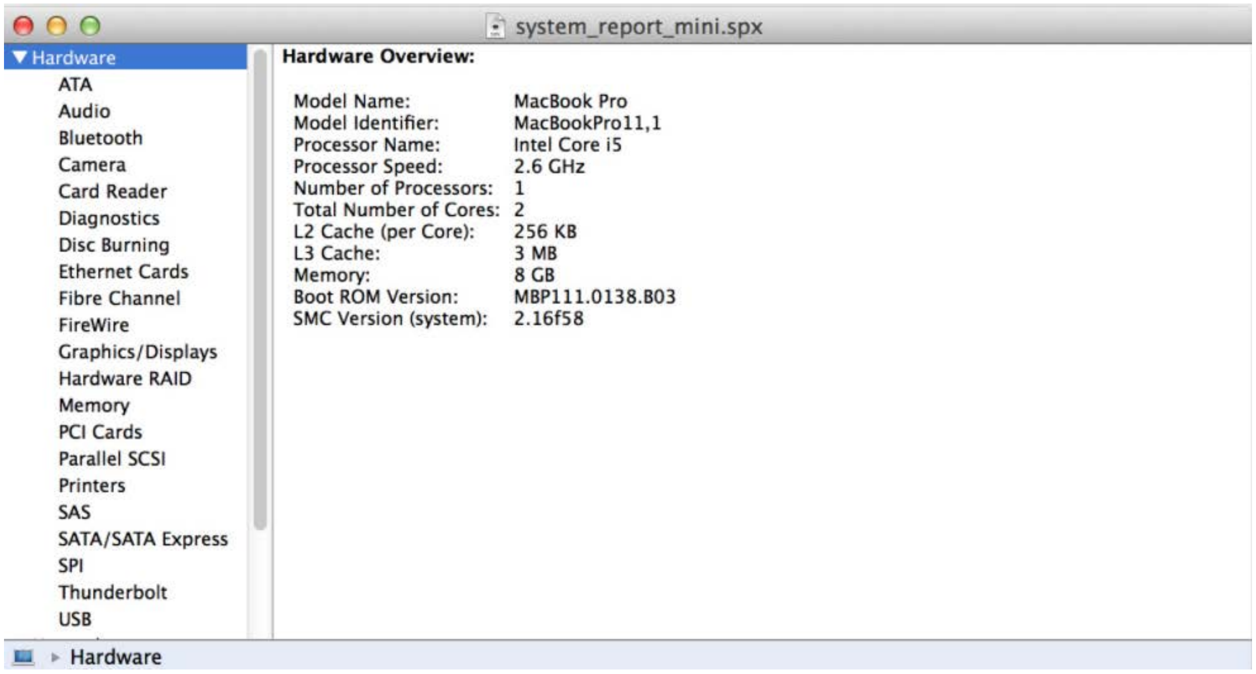
The NLP Suite automatically checks the memory size available on your computer when you run Stanford CoreNLP. But… you can check yourslf following these instructions. **8GB of memory may not be enough to run some of the more resourse-demanding annotators (e.g., parser) on large files.**

Mac

Via command line

Open command line and type:

```
System_profiler -detailLevel mini -xml > ~/Desktop/system_report_mini.spx
```



system_report_mini.spx

Hardware: ATA, Audio, Bluetooth, Camera, Card Reader, Diagnostics, Disc Burning, Ethernet Cards, Fibre Channel, FireWire, Graphics/Displays, Hardware RAID, Memory, PCI Cards, Parallel SCSI, Printers, SAS, SATA/SATA Express, SPI, Thunderbolt, USB

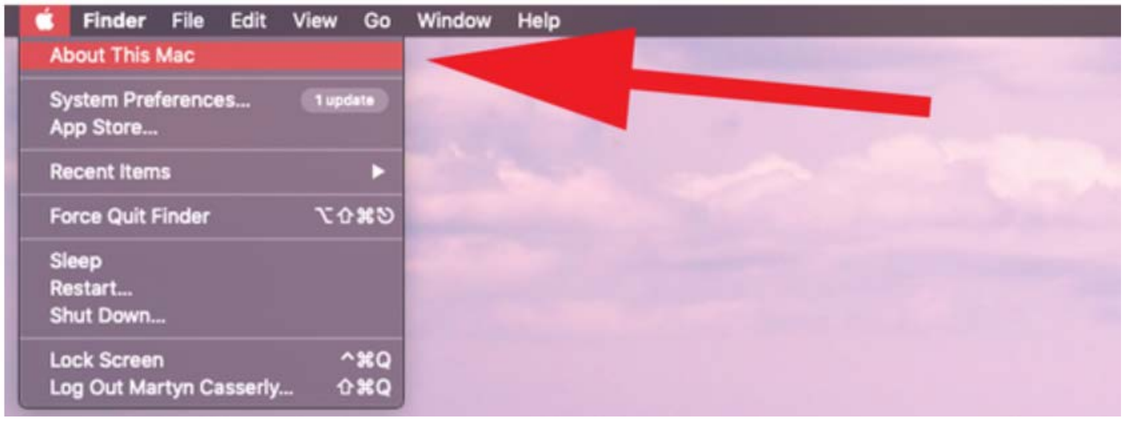
Hardware Overview:

```
Model Name:              MacBook Pro
Model Identifier:        MacBookPro11,1
Processor Name:          Intel Core i5
Processor Speed:         2.6 GHz
Number of Processors:    1
Total Number of Cores:   2
L2 Cache (per Core):     256 KB
L3 Cache:                3 MB
Memory:                  8 GB
Boot ROM Version:        MBP111.0138.B03
SMC Version (system):    2.16f58
```

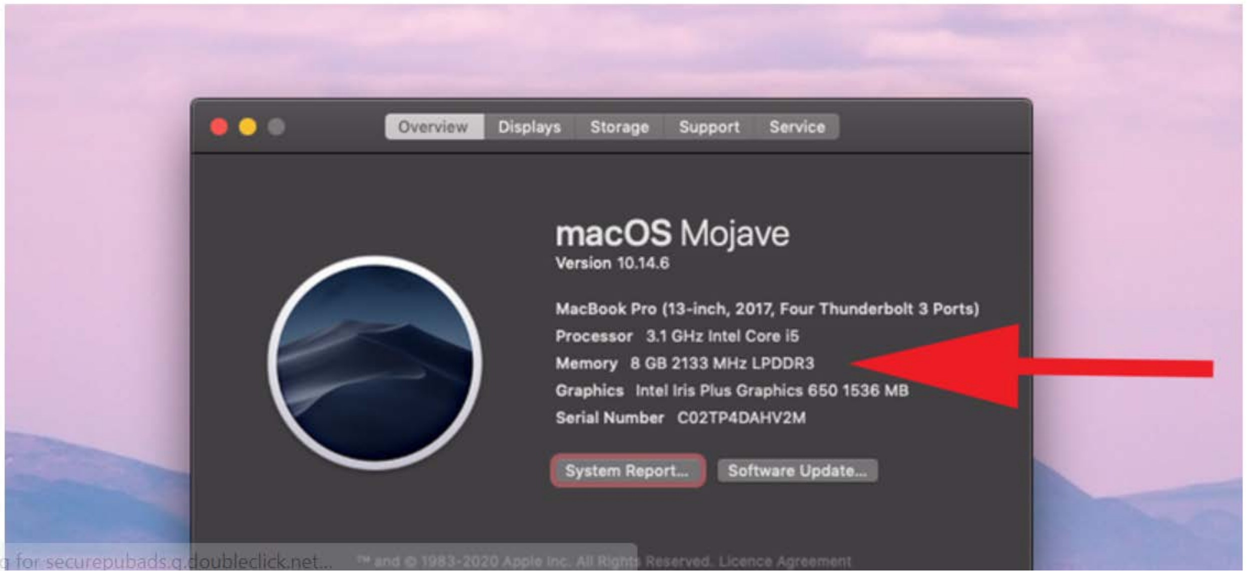
Hardware

Via About This Mac pane

1. Click on the Apple symbol in the top left corner of your screen.
2. Then select **About This Mac** from the menu that appears.



3. In the box that appears you'll see various details, including the installed version of macOS, model name, and the amount of Memory, which is another name for RAM. On our test MacBook Pro, as you can see from the image below, we have 8GB of RAM.



macOS Mojave
Version 10.14.6

MacBook Pro (13-inch, 2017, Four Thunderbolt 3 Ports)
Processor 3.1 GHz Intel Core i5
Memory 8 GB 2133 MHz LPDDR3
Graphics Intel Iris Plus Graphics 650 1536 MB
Serial Number C02TP4DAHV2M

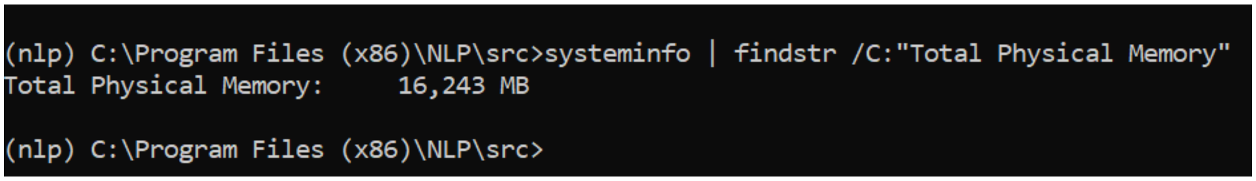
g for securepubads.g.doubleclick.net... ™ and © 1983-2020 Apple Inc. All Rights Reserved. Licence Agreement

Windows

Via command line

In command line type:

systeminfo | findstr /C:"Total Physical Memory"



```
(nlp) C:\Program Files (x86)\NLP\src>systeminfo | findstr /C:"Total Physical Memory"
Total Physical Memory:     16,243 MB

(nlp) C:\Program Files (x86)\NLP\src>
```

You have 16GB of memory, using the formula to convert 16,243 MB to GB:

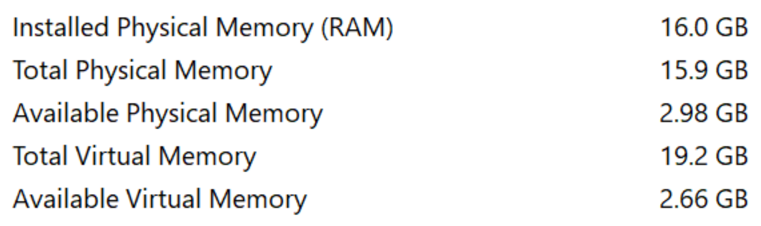
1 MB = 0.0009765625 GB

16,243 MB × 0.0009765625 GB = 15.86230469 GB

Via Windows Start menu

Click on the Windows Start menu and type in System Information. A list of search results pops up, among which is the System Information utility. Click on it. Scroll down to Installed Physical Memory (RAM) and see how much memory is installed on your computer.

You will see in a long list of items:



Installed Physical Memory (RAM)	16.0 GB
Total Physical Memory	15.9 GB
Available Physical Memory	2.98 GB
Total Virtual Memory	19.2 GB
Available Virtual Memory	2.66 GB

JAVA

JAVA TIPS UNDER CONSTRUCTION

https://stackoverflow.com/questions/12797560/command-line-tool-to-find-java-heap-size-and-memory-used-linux

https://mkyong.com/java/find-out-your-java-heap-memory-size/

1. how do find out Java memory allocation on a machine
2. how do we change Java memory settings on a machine

A 32bit JVM can only handle about 1.5 GB or RAM when starting it up, you need either 64-bit JVM or a different solution. You pass it on the command line, for instance: java -Xmx1024m

If you use a 64-bit JVM, you can set the heap size to -Xmx30G.

Use 64 bit JVM. Then java -Xmx12g