# 🚀在Python中自然语言处理速度提高100倍

我还出版了一本Jupyter笔记本，上面有我在这篇文章中描述的例子。

当我们去年发布Python时，我们得到了来自社区的一个惊人的反馈，人们开始将它用于许多应用程序，其中一些与我们最初的dialog用例非常不同。

我们发现，虽然对话信息的速度完全可以，但在较大的新闻文章中可能会非常慢。

我决定对此进行详细的调查，结果是比以前的版本（几千字/秒）快100倍，同时保持了同样的准确性，以及Python库的易用性和生态系统。

在这篇文章中，我想分享一些关于这个项目的经验教训，特别是：

* 如何在Python中设计高速模块，
* 如何利用spaCy的内部数据结构来高效地设计超快NLP函数。

所以我在这里有点作弊，因为我们会讨论Python，还有一些Cython魔术，但是，你知道吗？赛顿是，所以别让那吓跑你！

当前的Python程序已经是Cython程序。

有几种情况下，您可能需要这样的提速，例如：

* 您正在使用Python为NLP开发一个生产模块，
* 您正在使用Python计算大型NLP数据集的分析，
* 您正在为Pythorch/TensorFlow这样的深度学习框架预处理一个大型培训集，或者您的深度学习批处理加载器中有一个繁重的处理逻辑，这会减慢您的培训速度。

在我们开始之前的最后一件事是：我还出版了一本Jupyter笔记本，上面有我在这篇文章中谈到的工作示例。试试看！

### 火箭速度的第一步：剖面



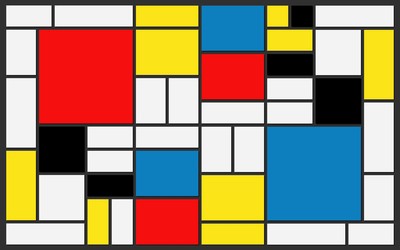
他首先要知道的是，在纯Python中，大多数代码可能都很好，但是如果你给它们一些爱，可能会有一些瓶颈函数会让你更快地得到数量级的代码。

因此，您应该从分析Python代码开始，找出慢部分的位置。一种选择是这样使用：

如果你使用神经网络，你可能会发现慢的部分是一些循环和一些Numpy数组操作（但我不会花时间在这里讨论Numpy，因为已经有一篇关于它的文章）。

那么，我们如何加速这些循环呢？

### 使用一点Cython的Python中的快速循环



让我们用一个简单的例子来解决这个问题。假设我们有一个大的矩形集，我们将其存储为Python对象的列表，例如矩形类的实例。我们模块的主要工作是遍历这个列表，以便计算有多少矩形的面积大于特定阈值。

我们的Python模块非常简单，如下所示：

检查矩形功能是我们的瓶颈！它在大量的Python对象上循环，这可能相当慢，因为Python解释器在每次迭代时都要做大量的工作（在类中查找area方法，打包和解包参数，调用Python API…）。

Cython来帮我们加速循环。

Cython语言是Python的超集，包含两种对象：

* Python对象是我们在常规Python中操作的对象，如数字、字符串、列表、类实例…
* Cython C对象是C或C++对象，如双、int、浮点、结构、向量，它们可以由Cython在超快速低级代码中编译。

快速循环只是Cython程序中的一个循环，在这个循环中我们只访问Cython C对象。

设计这样一个循环的直接方法是定义C结构，它将包含计算过程中需要的所有内容：在我们的例子中，是矩形的长度和宽度。

然后，我们可以将矩形列表存储在这样的结构的C数组中，以便传递给check\_rectangle函数。这个函数现在必须接受一个C数组作为输入，因此将通过使用C def关键字而不是def来定义Cython函数（注意cdef也用于定义cythonc对象）。

下面是我们的Python模块的快速Cython版本：

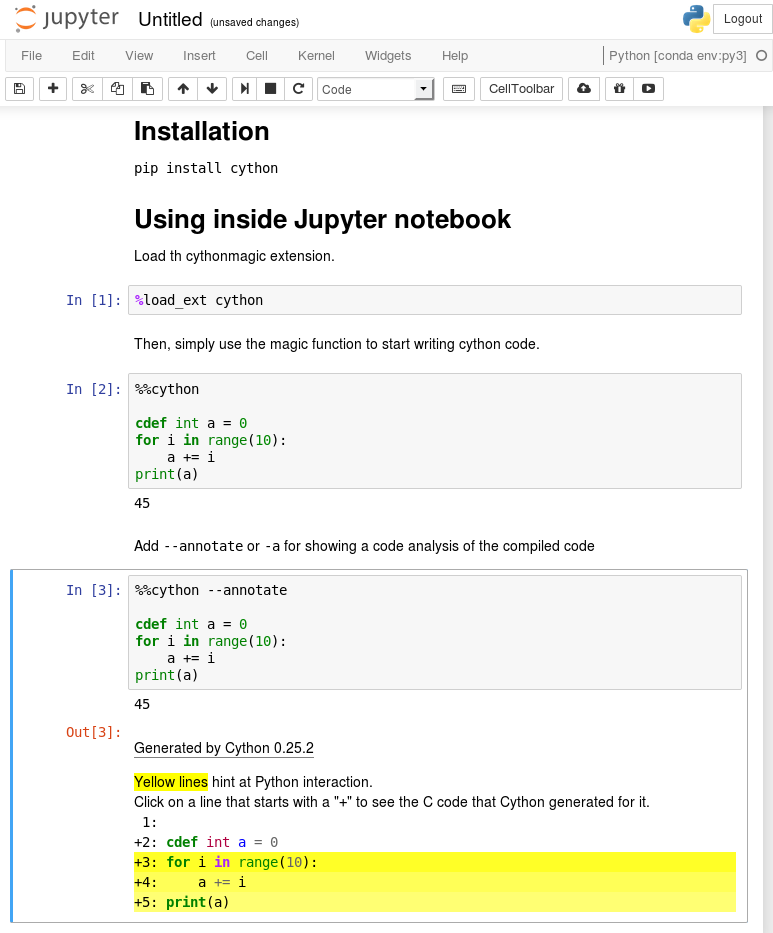
在这里，我们使用了一个C指针的原始数组，但您也可以选择其他选项，特别是。在这段代码中，我还使用了的方便的Pool（）内存管理对象，以避免手动释放已分配的C数组。当池被Python垃圾收集时，它会自动释放我们使用它分配的内存。

spaCy的API的Cython约定页面是NLP中Cython实际使用的一个很好的参考。

### 让我们试试那个代码！

这里有很多方法可以测试、编译和分发Cython代码！Cython甚至可以像Python一样使用。

先用pip安装Cython



#### 朱庇特的第一次测试

将Cython扩展加载到带有%Load\_ext Cython的Jupyter笔记本中。

现在您可以使用%%Cython编写类似Python代码的Cython代码。

如果在执行Cython单元格时出现编译错误，请确保检查Jupyter终端输出以查看完整消息。

大多数情况下，如果%Cython编译到C++（例如，如果您使用SPACE Cython API）或如果编译器抱怨NUMPY，则会丢失一个-+标签。

正如我在一开始提到的，检查这篇文章的附件，它包含了我们在Jupyter中讨论的所有示例。

#### 编写、使用和分发Cython代码

Cython代码是用.pyx文件编写的。这些文件由Cython编译器编译成C或C++文件，然后用系统的C编译器编译成字节码级。然后，Python解释器可以使用字节码级文件。

可以使用pyxPosit直接在python中加载一个.pyx文件：

您还可以将Cython代码构建为Python包，并将其作为常规Python包导入/分发（如详细说明）。这可能需要一些时间才能开始工作，特别是在所有平台上。如果你需要一个工作的例子，是一个相当全面的例子。

在我们讨论一些NLP之前，让我们快速讨论def、cdef和cpdef关键字，因为它们是开始使用Cython所需要掌握的主要内容。

在Cython程序中可以使用三种类型的函数：

* Python函数，用常用的关键字def定义。它们接受Python对象作为输入和输出。在内部，它们既可以使用Python和C/C++对象，也可以调用Cython和Python函数。
* 用cdef关键字定义的Cython函数。它们可以作为输入，在内部使用，同时输出Python和C/C++对象。这些函数不能从Python空间访问（即Python解释器和其他将导入Cython模块的纯Python模块），但它们可以由其他Cython模块导入。
* CpDEF关键字定义的Cython函数类似于CDEF Cython函数，但它们也提供了Python包装器，因此它们可以从Python空间调用（以Python对象作为输入和输出），也可以从其他Cython模块调用（以C/C++或Python对象作为输入）。

CDEF关键字还有另一个用途，就是在代码中键入Cython C/C++对象。除非使用此关键字键入对象，否则它们将被视为Python对象（因此访问速度慢）。

### 💫使用Cython和spaCy加速NLP

他的动作很快很好，但是…我们还是没有在这里做NLP！没有字符串操作，没有unicode编码，没有我们在自然语言处理中幸运的微妙之处。

官方Cython文档甚至使用了C级字符串：

一般来说：除非您知道自己在做什么，否则尽可能避免使用C字符串，而是使用Python字符串对象。

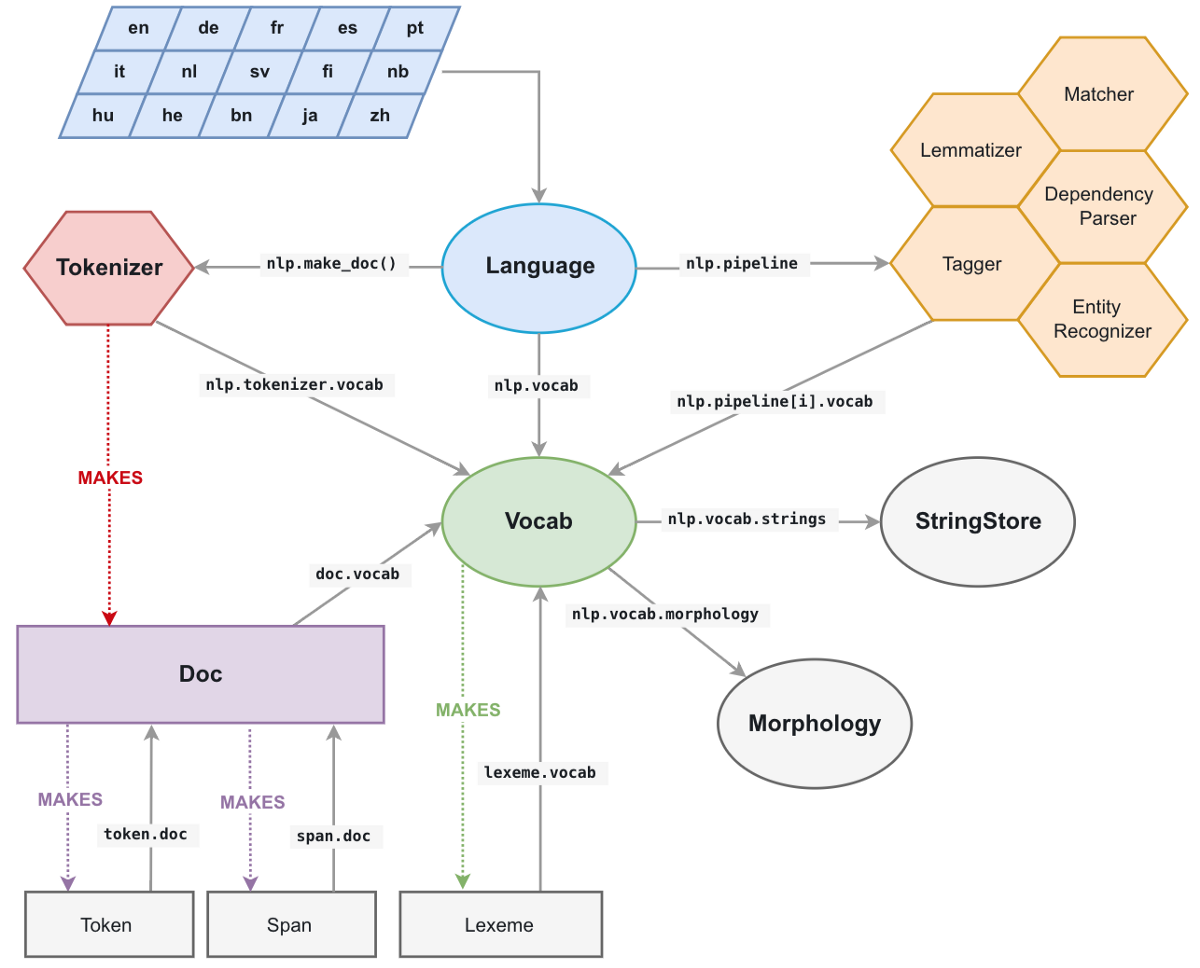
那么，当我们使用字符串时，如何在Cython中设计快速循环呢？

💫spaCy让我们得到了保护。

斯帕西解决这个问题的方法相当聪明。

#### 将所有字符串转换为64位哈希

spaCy中的所有unicode字符串（标记的文本、小写文本、引理形式、POS标记标签、解析树依赖标签、命名实体标记…）都存储在一个称为StringStore的数据结构中，在这个数据结构中它们被64位散列（即C级别）索引。



StringStore对象实现了Python unicode字符串和64位散列之间的查找。

它可以从spaCy中的任何位置和每个对象（见左侧）访问，例如nlp.vocab.strings、doc.vocab.strings或span.doc.vocab.string。

当模块需要对某些令牌执行快速处理时，它只需使用C级64位散列而不是字符串。调用StringStore查找表将返回与散列关联的Python unicode字符串。

但是spaCy做的不止这些，它还允许我们访问文档和词汇表的完全填充的C级结构，我们可以在Cython循环中使用这些结构，而不必构建自己的结构。

#### SpaCy的内部数据结构

与spaCy文档相关联的主要数据结构是doc object，它拥有已处理字符串的标记序列（“单词”）及其在名为doc.C的C级对象（标记结构数组）中的所有注释。

\*\*\*\*结构包含我们需要的关于每个令牌的所有信息。这些信息存储为64位散列，可以像我们刚才看到的那样重新关联到unicode字符串。

要确切地了解这些漂亮的C结构中的内容，只需看看新创建的spaCy💫。

让我们在一个简单的NLP处理示例中看到这一点。

### 🚀使用spaCy和Cython进行快速NLP处理

假设我们有一个需要分析的文本文档数据集。

在左边，我写了一个脚本，它构建了一个由spaCy解析的10个文档的列表，每个文档包含约170k个单词。我们也可以有170k个文档，每个文档包含10个单词（就像一个对话框数据集），但是创建起来比较慢，所以让我们坚持使用10个文档。

我们想在这个数据集上执行一些NLP任务。例如，我们想计算“run”一词在数据集中用作名词的次数（即，用spaCy标记的“NN”词性标记）。

这样做的Python循环既短又简单：

但它也相当慢！在我的笔记本电脑上，这段代码需要1.4秒才能得到答案。如果我们有一百万份文件，我们需要一天多的时间才能得到答案。

我们可以使用多处理，但因为你也必须处理，请注意Cython也可以！这甚至可能是Cython最好的部分，因为GIL被释放了，所以我们全速运行，Cython基本上直接调用引擎盖下的OpenMP。我没有时间在这里讨论并行性，所以请查看更多详细信息。

现在，让我们尝试使用spaCy和一些Cython来加速我们的Python代码。

首先，我们必须考虑数据结构。我们需要数据集的C级数组，其中有指向每个文档的TokenC数组的指针。我们还需要将使用的测试字符串（“run”和“NN”）转换为64位散列。

当处理所需的所有数据都在C级对象中时，我们就可以在数据集上以全C速度迭代。

下面是如何用Cython和spaCy编写这个示例：

代码有点长，因为在调用Cython函数之前，我们必须快速声明并填充main中的C结构。

但它也快得多！在我的Jupyter笔记本中，这个Cython代码运行大约需要20毫秒，是纯Python循环的80倍。

绝对速度对于一个写在Jupyter笔记本单元中的模块来说也是令人印象深刻的，它可以与其他Python模块和函数进行本机接口：在20毫秒内扫描约170万字意味着我们每秒处理高达8000万字。

这就结束了我们使用Cython进行NLP的快速介绍。希望你喜欢。

关于Cython还有很多其他的事情要说，但是这会让我们离简单的介绍太远了。从现在开始最好的地方可能是一个概述和NLP。

Don’t hesitate to give us a few claps 👏 if you want more content like that!

\*. [^](#c72f) If you use low level structures several times in your code, a more elegant option than populating C structures each time, is to design our Python code around the low level structures with [a Cython extension type](http://cython.readthedocs.io/en/latest/src/userguide/extension_types.html) wrapping the C level structures. This is how most of spaCy is structured and it is a very elegant way to combine fast speed, low memory use and the easiness of interfacing with external Python libraries and functions.