Article from

Predictive Analytics and Futurism

May 2019
Issue 20

Getting C++ Performance From Python With Cython

By Jeff Heaton

Python programs have a wealth of packages that can increase runtime performance. Packages such as Numpy, Scipy, TensorFlow and PySpark are just a few of those available to optimize your program's performance. Normally, when your Python code makes extensive use of loops that process mathematical equations, performance can suffer. Heavily nested mathematical oriented loops and multidimension arrays are normally the domain where compiled low-level languages such as C++ are best suited. However, by making use of a Python package called Cython, you can achieve performance close to that of C++ in Python.

> Cython works by transforming Python into C code. The output from Cython is literally a .C file.

Performance is not the only reason to consider using Cython. You can give the compiled binary produced by Cython to external users of your application, allowing some degree of protection of the intellectual property (IP). Cython works by transforming Python code into C code. The output from Cython is literally a .C file that you must compile with a C compiler. The code generated by compiling a C program to an executable is very difficult to decompile back into C, let alone further back into Python. This makes Cython an effective tool to protect IP contained in your Python source files. While no software protection scheme is perfect, it is much more difficult to reverse engineer compiled C code than higher-level languages, such as R and Python.

Because the output of Cython is C-source code, you must have a C compiler installed to make use of Cython. On Macintosh and Linux, this is easy. Both of these two platforms have open-source C compilers available. Windows is a bit more complex. For Windows, you will need one of the Visual C++ compilers or one of the open-source compilers. Docker is also an option for Windows, because it allows emulation of standard UNIX environments.

CYTHON FOR EXTENSIONS

This article will demonstrate several different ways to use Cython. You can use Cython with either Python 2.x or 3.x; however, the examples contained in this article will make use of Python 3.x. As of the time I wrote this article, the latest version of Python was Python 3.7. I also used the Anaconda release of Python on a Macintosh computer and the GCC C++ compiler version 8.2. Other environments should work; however, small modifications may be necessary.

Each of these examples can be found on my GitHub repository.[1] If performance is the goal in your use of Cython, then you will likely be using Cython to create a Python extension. This allows you to compile part of your Python program to a compiled Cython extension that many of your Python programs can use. The code inside of this extension will be compiled to C and be very efficient. You can import this extension into your Python script just like a regular package.

As an example, Listing 1 shows an extension I created that will calculate the standard deviation of a population.

Listing 1
Calculate the Standard Deviation of a Population (Calculate.pyx)

```
import math

def sdev(lst):
    # Mean
    sum = 0
    for x in lst:
        sum += x

    mean = sum / len(lst)

    # Standard deviation
    sum = 0
    for x in lst:
        delta = x - mean
        sum += delta ** 2

    return math.sqrt(sum / len(lst))
```

While Python contains built-in support for calculating the standard deviation of a population or sample, the above code shows how this can be done from simple algebraic operations. This is exactly the type of code that Cython can speed up. In this form, the above code is standard Python and could be used with

or without Cython. Even when compiling from pure Python, Cython gives considerable speed improvements.

You should save the above code to a filename such as "Calculate.pyx." The PYX extension designates this code as Cython. You must now compile this Python code to C code and then to a Python extension. This is accomplished by a Python build script that is often named "setup.py" and is shown in Listing 2.

Listing 2
Build the Extension (setup.py)

```
from distutils.core import setup
from Cython.Build import cythonize

setup(
  ext_modules = cythonize("Calculate.pyx")
)
```

You should execute the build script to actually compile the Cython extension with the following command:

```
python setup.py build_ext --inplace
```

The "build_ext" parameter indicates that you are building an extension. The "inplace" parameter designates that the extension should be placed in the current folder, as opposed to copied to a system directory. Once this program is run, it will create a .SO file under Mac/Linux or a DLL under Windows. This file is the actual Cython extension. Listing 3 shows how to actually make use of this extension.

Listing 3
Test the Shared Object (test.py)

```
import Calculate as c

print(c.sdev([1,2,3,4,5]))
```

This program simply imports the calculation Cython extension and then computes the standard deviation of the set [1, 2, 3, 4, 5]. The test script is executed with the following Python command:

```
python test.py
```

This will output the standard deviation of the vector.

CYTHON FOR STANDALONE EXECUTABLES

It is also possible to produce a standalone executable with Cython. This executable will be a .EXE file in a Windows operating system or executable file on Linux or Mac (these operating systems do not have a specific extension for executable files). To demonstrate this, we will use a classic Python "Hello World" program, as shown in Listing 4.

Listing 4:
Calculate the Standard Deviation of a Population (HelloWorld.pyx)

```
print("Hello World")
```

Unfortunately, the commands to build a standalone executable are a bit more complex than the previous example. The first step is to invoke Cython and convert the HelloWorld.pyx file to HelloWorld.c.

```
cython --embed -o HelloWorld.c HelloWorld.pyx
```

The next step is to compile the HelloWorld.c to a standalone file. This will require the use of your C++ compiler. The command that I used for GCC is as follows:

```
gcc -Os -I /Users/jheaton/miniconda3/
include/python3.6m/ -o HelloWorldEXE Hel-
loWorld.c -L/Users/jheaton/miniconda3/
lib -lpython3.6m -lpthread -lm -lutil -ldl
```

There are two important paths that are provided to GCC. The first is the path to the include files needed to compile. This includes Python.h. The second path is the location of the Python libraries that are needed to compile your Cython extension. This includes the primary Python library, named "python3.6m"; however, additional libraries should also be specified with more "-l" arguments.

The resulting file is executable from the command line.

CALLING PYTHON PACKAGES FROM CYTHON PROGRAMS

Python programs make use of a variety of packages. Often predictive modeling programs will use Scipy, Numpy, Scikit-Learn, TensorFlow and potentially many others. If you are going to make use of these packages, it is necessary for your Cython program to have access to them. These packages are dynamically linked, and your Cython program will not run without them. You will need each of them to be present to run the stand alone Cython executable. You can look up the file location of any Python package with a single Python command. For example, to find the location of Numpy, you would use the following command:

```
python -c 'import numpy; print(numpy.__file__)'
```

NEXT STEPS

This article provided a brief introduction to Cython. Using these techniques alone, you can considerably increase the speed of Python programs that need to use loops for their calculations. However, this is only the beginning. Cython also adds extensions to the Python programming language that you can use to further enhance the performance of your Python code. Such extensions include static typing and multithreading. ■

Jeff Heaton, Ph.D., is a vice president and data scientist for RGA Reinsurance Company in Chesterfield, Missouri. He can be reached at *jheaton@rgare.com.*

ENDNOTE

1 Heaton, Jeff. GitHub. https://github.com/jeffheaton/present/tree/master/SOA/paf-cython (Accessed March 5, 2019).