0091_numba

September 22, 2018

1 Speed up Python by 1,000 times or more using numba!

Functions written in pure Python or NumPy may be speeded up by using the numba library and using the decorator @jit before a function. This is especially useful for loops where Python will normally compile to machine code (the language the CPU understands) for each iteration of the loop. Using numba the loop is compiled into machine code just once the first time it is called.

Let's look at an example:

```
In [1]: from numba import jit
        import numpy as np
        import timeit

        # Define a function normally without using numba

        def test_without_numba():
            for i in np.arange(1000):
                x = i ** 0.5
                x *= 0.5

        # Define a function using numba jit. Using the argument nopython=True gives the
        # fastest possible run time, but will error if numba cannot precomplile all the
        # code. Using just @jit will allow the code to mix pre-compiled and normal code
        # but will not be as fast as possible

        @jit(nopython=True)
        def test_with_numba():
            for i in np.arange(1000):
                x = i ** 0.5
                x *= 0.5

        # Run functions first time without timing (compilation included in first run)
        test_without_numba()
        test_with_numba()

        # Time functions with timeit (100 repeats).
        # Multiply by 1000 to give milliseconds
        timed = timeit.timeit(stmt = test_without_numba, number=100) * 1000
        print ('Milliseconds without numba: %.3f' %timed)

        timed = timeit.timeit(stmt = test_with_numba, number=100) * 1000
        print ('Milliseconds with numba: %.3f' %timed)
```

```
Milliseconds without numba: 183.035
Milliseconds with numba: 0.026
```

We have a 7,000 fold increase in speed!!

Note: not all code will be speeded up by numba. Pandas for example are not helped by numba, and using numba will actually slow panda code down a little (because it looks for what can be pre-complied which takes time). So always test numba to see which functions it can speed up (and consider breaking larger functions down into smaller ones so that blocks that can use numba may be separated out).

If the default decorator @jit is used, with no other arguments, numba will allow a mix of code that can be pre-compiled with code that can't. For the fastest execution use @jit(nopython=True), but you may need to break your function down because this mode will error if parts of the function cannot be pre-compiled by numba.