Using C Extensions in Python

Who has had performance issues?

Why C Extensions?

- Bare metal speed
- In my case overall execution time sped up 117x
- Dropping down to C is how many high-performance Python libraries work (Numpy, UJson, etc)
- Caveat: C extensions are harder to maintain, utilize other enhancements first.
- We'll see an interesting performance corner case of Python and see how to solve it with C extensions.

Follow Along

https://github.com/mattfowler/PythonCExtensions

The MC10 Platform

The Biostamp RC

The MC10 Platform

- Python platform for biometric sensor feature extraction
- Computes everything from sleep cycles to knee ROM
- Utilized Numpy, Scikit learn and custom algorithms

The Pedometer Algorithm

- 30 minutes to analyze 1 hour of data
- Run on gyroscopic and accelerometer data
- Relied on computing running standard deviations that fed into the next standard deviation calculation

The Pedometer Algorithm

- Analyze one hour at a time
- Perform data clean up
- Compute standard deviation over sliding windows 50 to 100 elements at a time
- Fancy proprietary math happens
- Primary bottleneck was the calculation of standard deviation.

The Pedometer Algorithm

```python
def mean(lst):
    return sum(lst) / len(lst)


def standard_deviation(lst):
    m = mean(lst)
    variance = sum([(value - m) ** 2 for value in lst])
    return math.sqrt(variance / len(lst))
```

Numpy!

- We’re already using an amazing optimized C library
- np.std(lst)

Numpy :(

```
Python: 0.06 seconds
NumPy: 0.39 seconds
```

Numpy is over 6x slower, why? Let's look at source code.

https://github.com/numpy/numpy/blob/9d0225b800df3c2b0bffee9960df87b15527509c/numpy/core/src/multiarray/calculation.c#L361

Numpy :(

- There’s a ton of stuff happening in this code.
- Constant time factors often dominate, especially when the sample size is small.

Building a C Extension

```python
from distutils.core import setup, Extension

std_module = Extension('std', sources=['python_vs_c/std.cpp'])

setup(name='std_performance',
      version='1.0',
      description='Module for calculating standard deviation.',
      ext_modules=[std_module])
```

Parsing Python Method Calls

```cpp
static PyObject * std_standard_dev(PyObject *self, PyObject* args)
{
    PyObject* input;
    PyArg_ParseTuple(args, "O", &input);

    int size = PyList_Size(input);

    std::vector<double> list;
    list.resize(size);

    for(int i = 0; i < size; i++) {
        list[i] = PyFloat_AS_DOUBLE(PyList_GET_ITEM(input, i));
    }

   return PyFloat_FromDouble(standardDeviation(list));
}
```

C++ Standard Deviation

```cpp
double standardDeviation(std::vector<double> v)
{
    double sum = std::accumulate(v.begin(), v.end(), 1.0);
    double mean = sum / v.size();

    double squareSum = std::inner_product(v.begin(), v.end(), v.begin(), 0.0);
    return sqrt(squareSum / v.size() - mean * mean);
}
```

Defining Python Methods

```c
static PyMethodDef std_methods[] = {
  {"standard_dev", std_standard_dev, METH_VARARGS,
   "Return the standard deviation of a list."},
  {NULL,  NULL}   /* sentinel */
};

static struct PyModuleDef stdmodule = {
    PyModuleDef_HEAD_INIT,
    "std",   /* name of module */
    NULL, /* module documentation, may be NULL */
    -1,
    std_methods
};
```

Register Module with Python

```c
PyMODINIT_FUNC PyInit_std(void)
{
    return PyModule_Create(&stdmodule);
}
```

Add Main Entry Point

```c
int main(int argc, char **argv)
{
    wchar_t *program = Py_DecodeLocale(argv[0], NULL);
    if (program == NULL) {
        fprintf(stderr, "Fatal error: cannot decode argv[0]\n");
        exit(1);
    }

    /* Add a built-in module, before Py_Initialize */
    PyImport_AppendInittab("std", PyInit_std);

    /* Pass argv[0] to the Python interpreter */
    Py_SetProgramName(program);

    /* Initialize the Python interpreter.  Required. */
    Py_Initialize();

    PyMem_RawFree(program);
    return 0;
}
```

Reference Counting

- Reference counting keeps track of who is referencing a variable.
- Python will free memory when reference count of an object reaches zero.
- This can lead to inexplicable crashes or memory leaks.
- You need to read the docs to understand what to do.
- https://docs.python.org/3/c-api/exceptions.html#c.PyErr_NewException

Performance Demo

Using C Extensions

- Overall algorithm sped up 117x, runtime went from 30 minutes to under 30 seconds.
- C extensions are hard. Prioritize other optimizations first, correct usage of NumPy, Numba, etc.