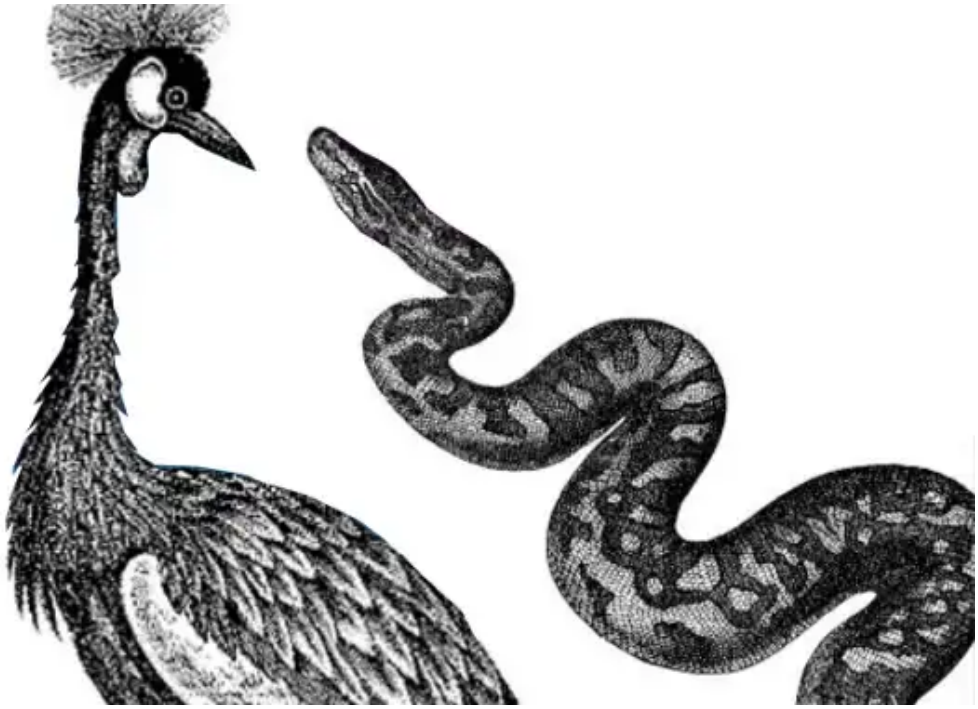
Published in SciSharp STACK

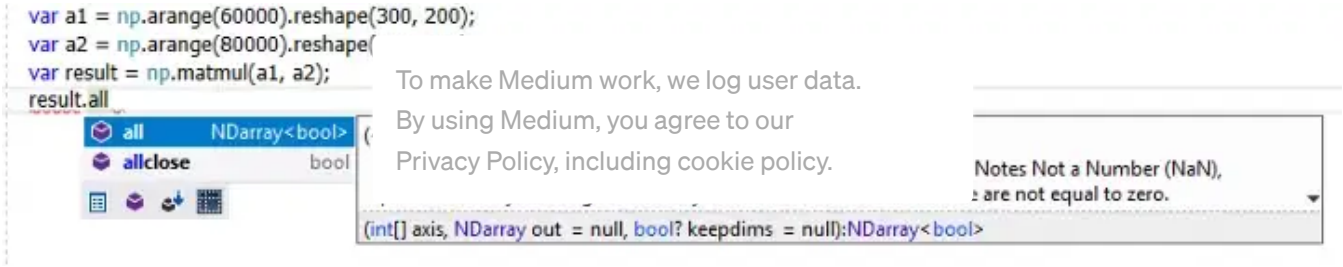
Meinrad Recheis Follow
Jun 5, 2019 · 4 min read · Listen

Using Python Libraries in .NET without a Python Installation

Thanks to the awesome pythonnet project we have a way to inter-operate between C# and Python. However, setting it up is problematic, as is deployment. Or is it?



In this article I introduce Python.Included which solves this problem elegantly and makes Python-interop fun and easy for .NET developers. As a proof-of-concept I present Numpy.NET which is a .NET Standard library that provides a strong typed API to Python's NumPy package and **does not require a local Python installation** on Windows.



Developers profit from the strong typed API of Numpy.NET which, contrary to a dynamic API, supports Visual Studio's Intellisense feature, showing the original NumPy documentation strings.

What is the problem?

Everyone has a different Python installation. Some still use Python 2.7, some use Python 3.5 or 3.6, and some are already on 3.7. When you use `pythonnet` it has to be compiled with different settings for every minor Python version and that version has to be installed for the code to run. So if you are working in a team, everyone is forced to have the same Python setup. With our SciSharp team, for instance, this is already not the case. If you want to deploy your .NET application on a machine you have to deploy Python first. From the perspective of a .NET developer it kind of sucks.

And yet, if you are working on Machine Learning and Artificial Intelligence, despite the efforts of Microsoft and SciSharp, at the moment you simply can not completely avoid Python. If you check out the list of projects using pythonnet it is evident that many companies in the AI field currently interface .NET with Python.

Python.Included to the rescue

But what if you could simply reference a Nuget package and everything is automatically set up correctly without additional manual tinkering? That was the vision that led me to create Python.Included which packages `python-3.7.3-embed-amd64.zip` in its assembly effectively allowing to reference Python via NuGet.

To prove that it works and to quickly provide all `numpy` -functionality that is still missing in NumSharp, I created Numpy.NET which is built upon Python.Included.

Proof-of-concept: Numpy.NET

Numpy.NET provides strong-typed wrapper functions for `numpy` , which means you don't need to use the `dynamic` keyword at all, but this is a rabbit hole to delve into in

another article. Instead we focus on how Numpy.NET uses Python.Included to auto-deploy Python on dema o call into it.

This is the setup code tl enes. You don't have to do any of this. As soon as you are using one of its functions, i.e.

```
var a = np.array(new [,] {{1, 2}, {3, 4}});,
```

`Numpy.dll` sets up the embedded Python distribution which will unpack from the assembly in your local home directory (only if not yet installed):

```
var installer = new Python.Included.Installer();
installer.SetupPython(force:false).Wait();
```

Next (if not yet done in a previous run) it unpacks the `numpy` pip wheel which is packed into `Numpy.dll` as embedded resource and installs it into the embedded Python installation.

```
installer.InstallWheel(typeof(NumPy).Assembly, "numpy-1.16.3-cp37-cp37m-win_amd64.whl").Wait();
```

Finally, the `pythonnet` runtime is initialized and `numpy` is imported for subsequent use.

```
PythonEngine.Initialize();
Py.Import("numpy");
```

All this is happening behind the scenes and the user of `Numpy.dll` does not have to worry about a local Python installation at all. In fact, even if you have installed any version of Python it won't matter.

Performance considerations

Pythonnet is known for being slow, so you might ask yourself if it is really a good idea to interface Python libraries with .NET using `pythonnet`. As always, it depends.

My measurements show that the overhead of calling into `numpy` with .NET compared to calling it directly from Python is about a factor 4. To be clear, this does not mean that Numpy.NET is four times slower than `numpy` in Python, it means that

there is an overhead of calling through pythonnet. Of course since Numpy.NET is calling `numpy`, the execu[illegible] is exactly the same.

Whether or not that ove[illegible]on the use case. If you are going back and forth between `CLR` and `Python` from within a nested loop you might have a problem. But mostly Python libraries are designed to be efficient and avoid looping over data. Numpy allows you to run an operation on millions of array elements with only one call. PyTorch and Tensorflow allow you to execute operations entirely on the GPU. So when used correctly, the interop-overhead will be negligible compared to the execution time of the operations when dealing with large amounts of data.

Roadmap

I know that there are a number of `numpy` ports for .NET, for instance via IronPython. But the IronPython project is still only supporting Python 2.7 and progressing very slowly. Libraries that depend on Python 3 are not available through IronPython and will not be in the near future.

My focus will be to make more Machine Learning and AI libraries available for .NET through pythonnet. The SciSharp team is also discussing ideas to make a faster version of pythonnet which avoids the use of the inherently slow DynamicObject.

Please try out Numpy.NET and let me know how it worked out for you. I'll be grateful for any comments or suggestions and I hope that my work will help the .NET Machine Learning community to grow and prosper.