Converting numpy dtypes to native python types

If I have a numpy dtype, how do I automatically convert it to its closest python data type? For example,

```
numpy.float32 -> "python float"
numpy.float64 -> "python float"
numpy.uint32  -> "python int"
numpy.int16   -> "python int"
```

I could try to come up with a mapping of all of these cases, but does numpy provide some automatic way of converting its dtypes into the closes possible native python types? This mapping need not be exhaustive, but it should convert the common dtypes that have a close python analog. I think this already happens somewhere in numpy.

python numpy

edited Nov 12 '13 at 12:26 asked Feb 26 '12 at 11:40

conradlee

1,514 2 22 46

6 Answers

Use either `a.item()` or `np.asscalar(a)` to convert most NumPy values to a native Python type:

```
import numpy as np
# examples using a.item()
type(np.float32(0).item()) # <type 'float'>
type(np.float64(0).item()) # <type 'float'>
type(np.uint32(0).item())  # <type 'long'>
# examples using np.asscalar(a)
type(np.asscalar(np.int16(0)))   # <type 'int'>
type(np.asscalar(np.cfloat(0)))  # <type 'complex'>
type(np.asscalar(np.datetime64(0)))  # <type 'datetime.datetime'>
type(np.asscalar(np.timedelta64(0))) # <type 'datetime.timedelta'>
...
```

Read more in the NumPy manual. For the curious, to build a table of conversions for your system:

```
for name in dir(np):
    obj = getattr(np, name)
    if hasattr(obj, 'dtype'):
        try:
            npn = obj(0)
            nat = npn.item()
            print('%s (%r) -> %s'%(name, npn.dtype.char, type(nat)))
        except:
            pass
```

There are a few NumPy types that have no native Python equivalent on some systems, including: `clongdouble`, `clongfloat`, `complex192`, `complex256`, `float128`, `longcomplex`, `longdouble` and `longfloat`. These need to be converted to their nearest NumPy equivalent before using `asscalar`.

edited Oct 5 '14 at 21:43

answered Jul 9 '12 at 6:27

Mike T
14.1k 5 41 73

How about:

```
In [51]: dict([(d, type(np.zeros(1,d).tolist()[0])) for d in
(np.float32,np.float64,np.uint32, np.int16)])
Out[51]:
{<type 'numpy.int16'>: <type 'int'>,
 <type 'numpy.uint32'>: <type 'long'>,
 <type 'numpy.float32'>: <type 'float'>,
 <type 'numpy.float64'>: <type 'float'>}
```

answered Feb 26 '12 at 12:55

unutbu
368k 46 665 791

I mention that type of solution as a possibility at the end of my question. But I'm looking for a systematic solution rather than a hard-coded one that just covers a few of the cases. For example, if numpy adds more dtypes in the future, your solution would break. So I'm not happy with that solution. – conradlee Feb 26 '12 at 13:51

The number of possible dtypes is unbounded. Consider `np.dtype('mint8')` for any positive integer `m`. There can not be an exhaustive mapping. (I also do not believe there is a builtin function to do this conversion for you. I could be wrong, but I don't think so :)) – unutbu Feb 26 '12 at 14:01

Python maps numpy dtypes to python types, I'm not sure how, but I'd like to use whatever method they do. I think this must happen to allow, for example, multiplication (and other operations) between numpy dtypes and python types. I guess their method does not exhaustively map all possible numpy types, but at least the most common ones where it makes sense. – conradlee Feb 26 '12 at 20:54

found myself having mixed set of numpy types and standard python. as all numpy types derive from `numpy.generic`, here's how you can convert everything to python standard types:

```
if isinstance(obj, numpy.generic):
    return numpy.asscalar(obj)
```

answered Apr 24 '13 at 10:46

tm_lv
2,297 2 14 14

I think you can just write general type convert function like so:

```
import numpy as np

def get_type_convert(np_type):
   convert_type = type(np.zeros(1,np_type).tolist()[0])
   return (np_type, convert_type)

print get_type_convert(np.float32)
>> (<type 'numpy.float32'>, <type 'float'>)

print get_type_convert(np.float64)
>> (<type 'numpy.float64'>, <type 'float'>)
```

This means there is no fixed lists and your code will scale with more types.

answered Feb 26 '12 at 18:45

Matt Alcock
3,452 3 20 48

Do you know where the source code is for the part of the tolist() method that maps numpy types to python types? I took a quick look but couldn't find it. – conradlee Feb 26 '12 at 22:15

This is a bit of a hack what I'm doing is generating a `numpy.ndarray` with 1 zero in it using `zeros()` and the calling the `ndarrays` `tolist()` function to convert into native types. Once in native types i ask for the type an return it. `tolist()` is a fucntion of the `ndarray` – Matt Alcock Feb 26 '12 at 22:27

Yeah I see that---it works for what I want and so I've accepted your solution. But I wonder how tolist() does its job of deciding what type to cast into, and I'm not sure how to find the source. – conradlee Feb 26 '12 at 22:35

numpy.sourceforge.net/numdoc/HTML/numdoc.htm#pgfId-36588 is where the function is documented. I thought inspect might be able to help find more information but no joy. Next step I tried to clone github.com/numpy/numpy.git and run `grep -r 'tolist' numpy .` (still in progress, numpy is massive!) – Matt Alcock Feb 26 '12 at 23:01

You can also call the `item()` method of the object you want to convert:

```
>>> from numpy import float32, uint32
>>> type(float32(0).item())
<type 'float'>
>>> type(uint32(0).item())
<type 'long'>
```

edited Mar 5 '14 at 20:47

Mike T
14.1k 5 41 73

answered Mar 5 '14 at 17:33

Aryeh Leib Taurog
2,007 16 29

---

numpy holds that information in a mapping exposed as `typeDict` so you could do something like the below::

```
>>> import __builtin__
>>> import numpy as np
>>> {v: k for k, v in np.typeDict.items() if k in dir(__builtin__)}
{numpy.object_: 'object',
 numpy.bool_: 'bool',
 numpy.string_: 'str',
 numpy.unicode_: 'unicode',
 numpy.int64: 'int',
 numpy.float64: 'float',
 numpy.complex128: 'complex'}
```

If you want the actual python types rather than their names, you can do ::

```
>>> {v: getattr(__builtin__, k) for k, v in np.typeDict.items() if k in vars(__builtin__)}
{numpy.object_: object,
 numpy.bool_: bool,
 numpy.string_: str,
 numpy.unicode_: unicode,
 numpy.int64: int,
 numpy.float64: float,
 numpy.complex128: complex}
```

edited Jan 21 at 9:55

answered Jan 21 at 9:08

Meitham
3,028 1 15 33