NumPy Primer

An introduction to numeric computing in Python

What is NumPy?

Numpy, SciPy and Matplotlib: MATLAB-like functionality for Python

Numpy:

- Typed multi-dimensional arrays
- Fast numerical computation
- High-level mathematical functions

Why do we need NumPy?

Numeric computing in Python is slow.

1000 x 1000 matrix multiply

Triple loop: > 1000 seconds

NumPy: 0.0279 seconds

Overview

1. Arrays

2. Shaping and transposition

3. Mathematical operations

4. Indexing and slicing

5. Broadcasting

Arrays

```
import numpy as np

a = np.array([[1,2,3],[4,5,6]], dtype=np.float32)

print a.ndim, a.shape, a.dtype
```

1. Arrays can have any number of dimensions, including zero (a scalar).

2. Arrays are typed. Common dtypes are: np.uint8 (byte), np.int64 (signed 64-bit integer), np.float32 (single-precision float), np.float64 (double-precision float).

3. Arrays are dense. Each element of the array exists and has the same type.

Arrays, creation

1. np.ones, np.zeros
2. np.arange
3. np.concatenate
4. np.astype
5. np.zeros_like, np.ones_like
6. np.random.random

Arrays, failure cases

1. Must be dense, no holes

2. Cannot mix type

3. Cannot combine arrays of different shape

Shaping

```
a = a.reshape(3,2)
```

```
a = a.reshape(-1,2)
```

```
a = a.ravel()
```

1. Total number of elements cannot change

2. Use -1 on an axis to automatically infer shape

3. Note: default order is row-major, MATLAB is column-major

Return values

NumPy operations return views or copies.

Views share the underlying storage of the original array. Changing the values of a view will change the original and vice versa.

Read the documentation to determine if an operation returns a copy or a view. Most operations return a view when possible and a copy otherwise.

np.copy, np.view will make explicit copies and views.

Transposition

```
a = np.arange(10).reshape(5,2)
```

```
a = a.T
```

```
a = a.transpose((1,0))
```

np.transpose will permute axes.

Terminology: Most operations have an np and ndarray equivalent.

```
a = np.transpose(a,(1,0))
```

Saving and loading arrays

```
np.savez('data.npz', a=a)

data = np.load('data.npz')

a = data['a']
```

1. NPZ file can hold multiple arrays

2. np.savez_compressed is an alternative

Image arrays

Images are 3D: width, height and channels (typically R, G, B)

Common image formats:

height x width x RGB (band-interleaved)

RGB x height x width (band-sequential)

Alternatives: Channel order may be RGB or BGR, spatial dimensions may be transposed (width x height).

Saving and loading images

SciPy: skimage.io.imread, skimage.io.imsave

height x width x RGB

PIL / Pillow: PIL.Image.open, Image.save

width x height x RGB

OpenCV2: cv2.imread, cv2.imwrite

height x width x BGR

Image Examples

```
# convert height x width x RGB -> RGB x height x width

I = I.transpose((2,0,1))

# collapsing spatial dimensions

I = I.reshape(3,-1)

# convert RGB <-> BGR

I = I[::-1]
```

Recap

We know how to create arrays, reshape them and permute their axes.

Next, math on arrays

Mathematical operations

Arithmetic operations (e.g., add, subtract, multiply, divide and power) are element-wise.

Logical operations (e.g., `a<0`) return an array with dtype np.bool. More on this later.

`np.dot(a,b)` or `a.dot(b)` to form the matrix product.

In-place operations modify the array

```
a = a + b
```

```
a += b
```

Math, upcasting

The result will have a dtype that is more general or precise when the operands have differing type.

Failure case: upcast will not occur to prevent underflow or overflow. You must manually upcast one of the operands first.

Important note: images are typically stored as uint8. You will want to convert images to float32 or float64 before doing math on them.

```
img = skimage.io.imread('foo.jpg')

I = img/255.0 # 255.0 is float so result will be upcast
```

Math, universal functions

Terminology: a universal function is called a ufunc

Like arithmetic, universal functions are element-wise.

Examples: np.exp, np.sqrt, np.sin, np.cos, np.isnan

Indexing

```
x[0,0]        # top-left element

x[0,-1]       # first row, last column

x[0]          # first row

x[:,0]        # first column
```

1. Indices are zero-based

2. Multi-dimensional indices are comma separated (i.e., a tuple)

Indexing, slices and arrays

```
I[1:-1,1:-1]          # select all but one-pixel border

I = I[:,:,::-1]       # swap channel order

I[I<10] = 0           # set dark pixels to black

I[[1,3]]              # select 2nd and 4th row
```

1. A slice is a view, so you can write to it and the original array will be modified

2. Arrays can also be indexed by a list or another boolean array

Python slicing

Syntax: start:stop:step

```
a = list(range(10))

a[:3]                 # indices 0, 1, 2

a[-3:]                # indices 7, 8, 9

a[3:8:2]              # indices 3, 5, 7

a[4:1:-1]             # indices 4, 3, 2 (this one is tricky)
```

Axes

```
a.sum()                        # sum all entries

a.sum(axis=0)                  # sum over rows

a.sum(axis=1)                  # sum over columns

a.sum(axis=1, keepdims=True)
```

1. Use the axis parameter to control which axis NumPy operates on

2. Typically, the axis specified will disappear, keepdims keeps all dimensions

Broadcasting

```
a  = a + 1 # add one to every element
```

When operating on multiple arrays, broadcasting rules are used.

Each dimension must match, from right-to-left

1. Dimensions of size 1 will broadcast (as if the value was repeated).

2. Otherwise, the dimension must have the same shape.

3. Extra dimensions of size 1 are added to the left as needed.

Broadcasting example

Suppose we want to add a color value to an image

a.shape is 100, 200, 3

b.shape is 3

a + b will pad b with two extra dimensions so it has an effective shape of 1 x 1 x 3.

So, the addition will broadcast over the first and second dimensions.

Broadcasting failures

If a.shape is 100, 200, 3 but b.shape is 4 then a + b will fail. The trailing dimensions must have the same shape (or be 1).

Average image

Who is this?

Practice exercise (optional, not graded)

Compute the average image of faces.

1. Download Labeled Faces in the Wild dataset (google: LFW face dataset). Pick a face with at least 100 images.

2. Call numpy.zeros to create a 250 x 250 x 3 float64 tensor to hold the result

3. Read each image with skimage.io.imread, convert to float and accumulate

4. Write the averaged result with skimage.io.imsave