Python
Numpy

MACbioIDi – February – March 2018

Vectorization

- Programming style
  - Operations are applied to whole arrays instead of individual elements
- It allows to take advantage of the current processors
  - 128 bits registers
  - SIMD instructions
  - Multi-threading
- Some architectures based in SIMD works with this programming style:
  - GPU
  - FPGA

Programming languages

- Modern programming languages that support vectorization are commonly used in scientific settings:
  - Octave
  - R
  - Fortran
  - Matlab
  - **Python using NumPy Extension...**

NumPy Basics

- NumPy's main object is the homogeneous multidimensional array
  - Table of elements (usually numbers)
- In NumPy nomenclature:
  - Dimensions are called **axes**
  - Number of axes is called **rank**

```python
import numpy as np

oneDimArray = np.array([1,2,3,4])
twoDimArray = np.array([[1,2,3,4],[5,6,7,8]])
```

NumPy Basics

- It is very usual to detail the type of elements during ndarray construction
  - It is not mandatory but it is recommended

```
>> a = np.array([[1, 2, 3],
              [4, 5, 6]], float)
>> a.dtype
dtype('float64')

>> a = np.array([[1, 2, 3],
              [4, 5, 6]])
>> a.dtype
dtype('int64')
```

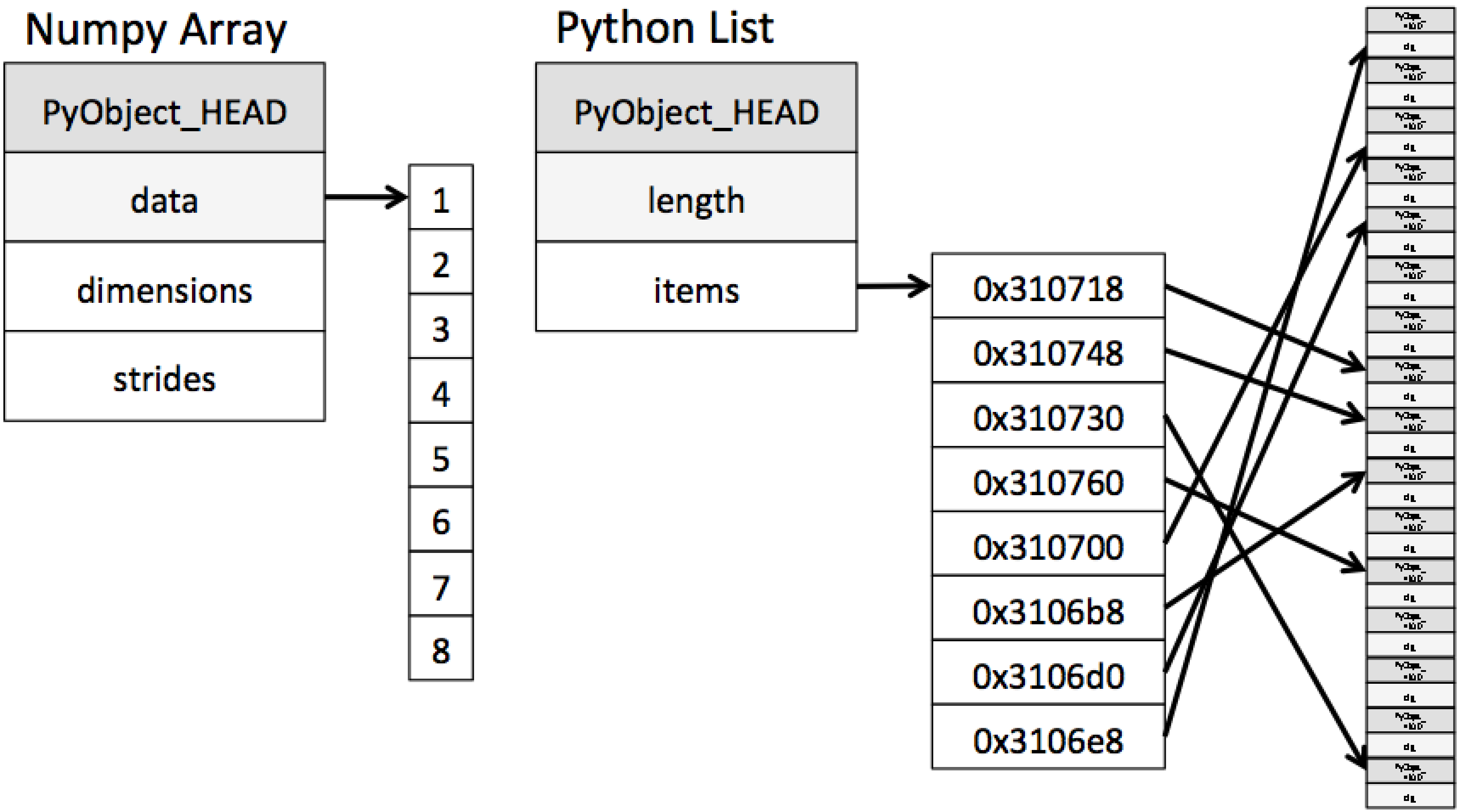
- Intrinsic NumPy array creation:

```
>> np.zeros((2,3))
>> np.identity(3)
>> np.arange(10)
>> np.arange(2,10)
>> np.arange(2,3,0.1)
>> np.linspace(1., 4., 6)
>> np.indices((3,3))
```

NumPy Basics

- NumPy's array class is called **ndarray**
  - **numpy.array** is a alias of this class
- Attributes:
  - ndarray.**ndim**
  - ndarray.**shape**
  - ndarray.**size**
  - ndarray.**dtype**
  - ndarray.**itemsize**
  - ndarray.**data**
  - ndarray.**strides**

NumPy’s array



Reference: https://jakevdp.github.io/blog/2014/05/09/why-python-is-slow/

NumPy.ndarray Functions

- ndarray.item([index])
- ndarray.itemset([index, value])
- ndarray.argmax([axis, out])
- ndarray.min([axis, out, keepdims])
- ndarray.clip([min, max, out])
- ndarray.transpose()
- ndarray.conj()
- ndarray.sum([axis, dtype, out, keepdims])
- ndarray.mean([axis, dtype, out, keepdims])
- ndarray.var([axis, dtype, out, ddof, keepdims])...

Reference: https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.ndarray.html

NumPy vector details

- Prest attention to the array's rank
  - You can get subtle bugs
- Column vector times row vector example:
  - Result must be a 5x5 matrix but… What is it happening?

```
a = np.random.rand(5)
print(np.dot(a,a.T))

>> [random scalar value]
```

*Check array's rank

NumPy vector details

- Let me try...

```
a = np.random.rand(5,1)
print(np.dot(a,a.T))

>> [random 5x5 matrix]
```

- What was it happening?
  - The first example contains a rank 1 array
  - You cannot transpose a rank 1 array
- Conclusion:
  - Don’t use 1 rank array or be very carefully

Indexing

- NumPy offers several ways to index into arrays:
  - **Slicing**

```
a = np.array([[1,2,3,4], [5,6,7,8],
        [9,10,11,12]])
>> print(a[:2, 1:3])
```

  - **Integer array indexing**

```
>> print(a[[0,1,2],[0,1,3]])
>> print(nb.array([a[0,0], a[1,1], [2,3]))
```

  - **Bolean array indexing**

```
boolIdx = (a > 2)
>> print(boolIdx)
```

Broadcasting

- This term describes how numpy treats arrays with different shapes during arithmetic operations
  - The smaller array is “broadcast” across the larger array so that they have compatible shapes

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + 100 \quad \rightarrow \quad \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 100 & 100 & 100 \end{bmatrix}$$

Broadcasting Example

Calories from Carbs, Proteins, Fats in 100g of different foods:

	Apples	Beef	Eggs	Potatoes
Carb	56	0	4.4	68
Protein	1.2	104	52	8
Fat	1.8	135	99	0.9

Broadcasting Example

```
A = np.array([[56, 0, 4.4, 68],
         [1.2, 104, 52, 8],
         [1.8, 135, 99, 0.9]],
         float)
print(A)

cal = A.sum(axis=0)
print("A shape = " + str(A.shape))
print("Cal shape = " +
str(cal.shape))

percentage1 =
100*A/cal.reshape(1,4)
percentage2 = 100*A/cal

print(percentage1)
print(percentage1)
```

Final Exercise

- Vectorizing Logistic Regression
  - With 'n' training samples: $\{(x^1 y^1)(x^2 y^2)\cdots(x^n y^n)\}$

$$Z = W^T X + b$$

where:

$$Z = [\hat{y}_1 \hat{y}_2 \cdots \hat{y}_n] \qquad W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_f \end{bmatrix} \qquad X = \begin{bmatrix} \vdots & \vdots & & \vdots \\ x_1 & x_2 & \cdots & x_n \\ \vdots & \vdots & & \vdots \end{bmatrix} \qquad b = [b_1 b_2 \cdots b_n]$$

  - Cost function:

$$L(\hat{y}, y) = -(y * \log \hat{y} + (1 - y) * \log(1 - \hat{y}))$$

Final Exercise

- In a Git repository, we have a template of Logistic Regression's Gradient Computation
  - Try to implement the **unfinished code**

Python
Numpy

MACbioIDi – February – March 2018