



numpy



覃雄派



提纲

- Numpy简介
- Numpy基本概念
- Numpy实例



numpy



numpy

- Numpy简介: What is Numpy?
 - Numpy, SciPy, and Matplotlib provide MATLAB-like functionality in python.
- Numpy Features:
 - Typed multidimensional **arrays** (matrices)
 - Fast numerical **computations** (matrix math)
 - High-level **math functions**



numpy

- Numpy简介: Why do we need Numpy
- Python does numerical computations slowly
- 1000 x 1000 matrix multiply
 - Python triple loop takes > 10 min.
 - Numpy takes ~0.03 seconds



numpy

- Functions of numpy
 - Arrays
 - Shaping and transposition
 - Mathematical Operations
 - Indexing and slicing
 - Broadcasting



numpy

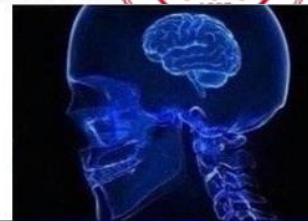


numpy

- Numpy基本概念
- Arrays: Structured lists of numbers.
 - Vectors
 - Matrices
 - Images
 - Tensors
 - ConvNets

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

MATRICES



IMAGES



TENSORS



CONVNETS

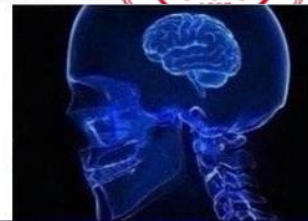


numpy

- Numpy基本概念
- Arrays: Structured lists of numbers.
 - Vectors
 - Matrices
 - Images
 - Tensors
 - ConvNets

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

MATRICES



IMAGES



TENSORS



CONVNETS

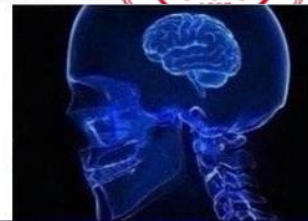


numpy

- Numpy基本概念
- Arrays: Structured lists of numbers.
 - Vectors
 - Matrices
 - Images
 - Tensors
 - ConvNets



MATRICES



IMAGES



TENSORS

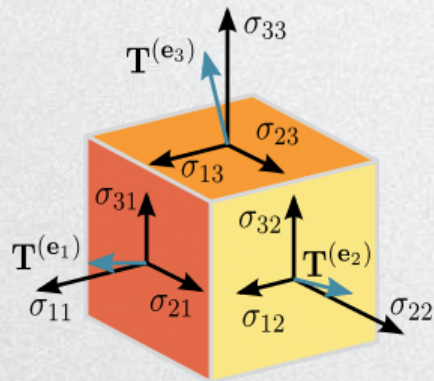


CONVNETS

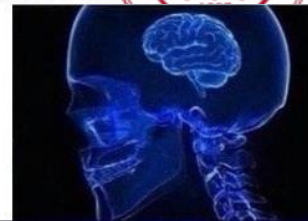


numpy

- Numpy基本概念
- Arrays: Structured lists of numbers.
 - Vectors
 - Matrices
 - Images
 - **Tensors**
 - ConvNets



MATRICES



IMAGES



TENSORS

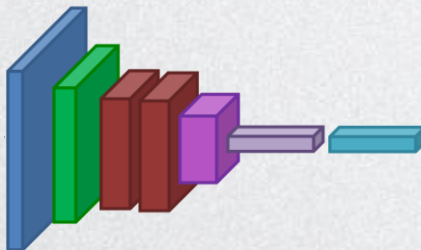


CONVNETS

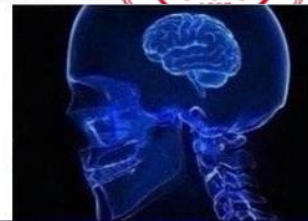


numpy

- Numpy基本概念
- Arrays: Structured lists of numbers.
 - Vectors
 - Matrices
 - Images
 - Tensors
 - ConvNets



MATRICES



IMAGES



TENSORS



CONVNETS





numpy



numpy

- Numpy实例
 - 创建1维和2维数组

```
import numpy
import numpy as np

print (np.array([1,2,3,4]) )# 创建1维数组
print (np.array([1,2,3,4], dtype=numpy.float64) )# 创建1维数组, 指定数据类型
print (np.array([1.2,2,3,4]) )# 创建1维数组

print (np.array([[1,2],[3,4]]) )# 创建2维数组
```

```
[1 2 3 4]
[1.  2.  3.  4.]
[1.2 2.   3.   4. ]
[[1 2]
 [3 4]]
```

numpy

- Numpy实例
 - 通过arange和linspace创建数组

```
print (np.arange(15))# 创建1维数组, 内容为[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
```

```
print (np.linspace(1,3,9))# 创建一个1维数组, 内容为[1, 3]之间的9个均匀间隔的数值[1.  1.25  1.5  1.75  2.  2.25  2.5  2.75  3.]
```

```
[1.   1.25 1.5  1.75 2.   2.25 2.5  2.75 3. ]
```

numpy

- Numpy实例
 - 全0、全1、单位矩阵

```
print (np.zeros((3,4)) )# 3行4列的矩阵, 元素都是0
print (np.ones((3,4)) ) # 3行4列的矩阵, 元素都是1
print (np.eye(3))# 3行3列的单位矩阵

print (np.zeros((2,2,2)) )# 2行2列2层的张量, 即3维数组, 元素都是0
```

```
[[0.  0.  0.  0.]
 [0.  0.  0.  0.]
 [0.  0.  0.  0.]]

[[1.  1.  1.  1.]
 [1.  1.  1.  1.]
 [1.  1.  1.  1.]]

[[1.  0.  0.]
 [0.  1.  0.]
 [0.  0.  1.]]

[[[0.  0.]
  [0.  0.]]

 [[0.  0.]
  [0.  0.]]]
```



numpy

- Numpy实例
 - random矩阵

```
# random random  
a = np.random.random((10,3))  
print(a)
```

```
[[0.99294316 0.64164652 0.74508153]  
 [0.14256115 0.28379451 0.71494861]  
 [0.47819953 0.57213991 0.85432021]  
 [0.68229682 0.30149229 0.75411727]  
 [0.18049368 0.77827893 0.53392431]  
 [0.53981262 0.38707349 0.2009222 ]  
 [0.79698328 0.51150558 0.41391815]  
 [0.62769333 0.88913342 0.84572052]  
 [0.36855129 0.24882605 0.1437408 ]  
 [0.98474505 0.22020911 0.35715525]]
```


numpy

- concatenate

```
a = np.array( [[11, 2], [5, 33]])  
b = np.array( [[34, 5], [77, 92]])  
print(a)  
print(b)
```

```
c = np.concatenate([a, b])  
print(c)
```

```
d = np.concatenate([a, b], axis=1)  
print(d)
```

```
[[11  2]  
 [ 5 33]  
 [34  5]  
 [77 92]]
```

```
[[11  2]  
 [ 5 33]  
 [34  5]  
 [77 92]]
```

按行拼接

```
[[11  2 34  5]  
 [ 5 33 77 92]]
```

按列拼接

numpy

- Numpy实例
 - zeros_like, ones_like矩阵

```
a = np.random.random((5, 3))  
b = np.zeros_like(a)  
print(b)  
c = np.ones_like(a)  
print(c)
```

```
[[0.  0.  0.]  
 [0.  0.  0.]  
 [0.  0.  0.]  
 [0.  0.  0.]  
 [0.  0.  0.]]  
[[1.  1.  1.]  
 [1.  1.  1.]  
 [1.  1.  1.]  
 [1.  1.  1.]  
 [1.  1.  1.]]
```

numpy

- Numpy实例
 - astype矩阵

```
# astype  
a = np.random.random((5,3))  
a = a+3  
print(a)  
a = a.astype( np.uint32)  
print(a)
```

```
[[3.78855597 3.52076375 3.54510472]  
 [3.46129928 3.5575374 3.70342108]  
 [3.38203519 3.45268724 3.03687321]  
 [3.00055904 3.41857347 3.37152952]  
 [3.63758365 3.33651026 3.27828937]]  
[[3 3 3]  
 [3 3 3]  
 [3 3 3]  
 [3 3 3]  
 [3 3 3]]
```

numpy

- Numpy实例
 - 显示数组的属性

```
ar = np.zeros((2,2,2)) # 2行2列2层的张量, 即3维数组, 元素都是0
print (ar.ndim)      # 数组的维数, 3
print (ar.shape)     # 数组每一维的大小, (2, 2, 2)
print (ar.size)      # 数组的元素个数, 8
print (ar.dtype)     # 元素类型float64
print (ar.itemsize ) # 每个元素所占的字节数, 8
```

```
3
(2, 2, 2)
8
float64
8
```

- Arrays can have any number of dimensions, including zero (a scalar).
- Arrays are typed: np.uint8, np.int64, np.float32, np.float64
- Arrays are dense. Each element of the array exists and has the same type.

numpy

- Numpy实例
 - Accessing the array

```
ar = (np.array( [[2,3,4],[5,6,7]] ))# 2行3列数组, 也就是一个矩阵
print (ar) # 输出整个数组
print (ar[1,2])# 输出一个元素, 7
print (ar[1,:])# 输出一行, 下标为1的行, 即[5 6 7]
print (ar[:,1])# 输出一列, 下标为1的列, 即[3 6]
print (ar[0:2,0:2])# 输出一个行列子集, 行下标为0、1, 列下标为0、1
ar[1,:] = [8,9,10]# 改变一行
print (ar)
```

} 注意代码后的注释

```
[[2 3 4]
 [5 6 7]]
7
[5 6 7]
[3 6]
[[2 3]
 [5 6]]
[[ 2  3  4]
 [ 8  9 10]]
```

numpy

- Python Slicing
 - Syntax: start:stop:step

```
#python list slicing
a = list(range(10))
print(a)
print(a[:3]) # indices 0, 1, 2
print(a[-3:]) # indices 7, 8, 9
print(a[3:8:2]) # indices 3, 5, 7
               #从3开始, 步长为2, 包头不包尾
print(a[4:1:-1]) # indices 4, 3, 2 (this one is tricky)
               #从4开始逆序, 步长为1, 包头不包尾
```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

[0, 1, 2]

[7, 8, 9]

[3, 5, 7]

[4, 3, 2]

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

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

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

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

#从3开始, 步长为2, 包头不包尾, 即不包括下标8

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

#从4开始逆序, 步长为1, 包头不包尾, 即不包括下标1

numpy

- Numpy实例

- Accessing the array: index

- `x[0,0]` # top-left element
 - `x[0,-1]` # first row, last column
 - `x[0,:]` # first row (many entries)
 - `x[:,0]` # first column (many entries)

- Notes:

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

numpy

- 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





Slices are **views**. Writing to a slice overwrites the original array.
Can also index by a list or Boolean array.

Swap channel order请参考，此处不展开，不做要求

<https://stackoverflow.com/questions/54951686/swapping-data-from-one-channel-to-another-using-numpy>

numpy

- 数据切片总结
 - 切片方式有2种
 - 1.与Python的列表类型类似，可以通过索引进行切片

	Expression	Shape
	<code>arr[:2, 1:]</code>	<code>(2, 2)</code>
	<code>arr[2]</code>	<code>(3,)</code>
	<code>arr[2, :]</code>	<code>(3,)</code>
	<code>arr[2:, :]</code>	<code>(1, 3)</code>
	<code>arr[:, :2]</code>	<code>(3, 2)</code>
	<code>arr[1, :2]</code>	<code>(2,)</code>
	<code>arr[1:2, :2]</code>	<code>(1, 2)</code>

numpy

- 数据切片总结
 - 切片方式有2种
 - 2.布尔型索引，即通过布尔型数组对数据进行切片

```
names = np.array(['Bob', 'Joe', 'Bob', 'Alice'])
data = np.random.randn(4,5)
print(names=='Bob') # names=='Bob'会返回一个布尔型的数组
print(data[names == 'Bob']) # 通过布尔型的数组进行切片
print(names!='Bob') # names!='Bob'会返回一个布尔型的数组
print(data[names != 'Bob']) # 通过布尔型的数组进行切片
print((names=='Bob') | (names=='Alice')) # 可以考虑多个条件
print(data[(names=='Bob') | (names=='Alice')]) # 通过布尔型的数组进行切片

data[data<0] = 0 # 通过布尔型切片选出所有的负数，统一置为0
print(data)
```

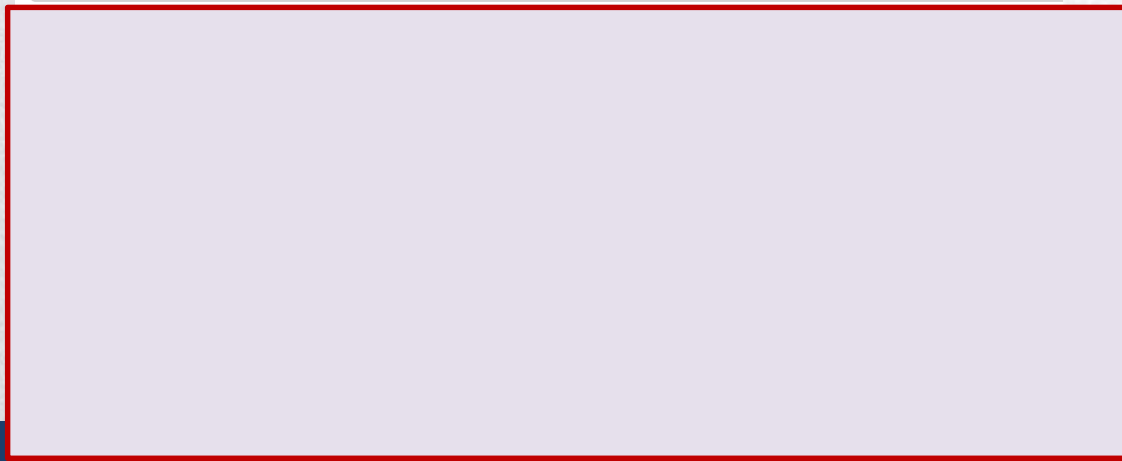
```
[ True False  True False]
[[ 1.63691799 -0.53269813  0.01980702  0.13260588 -2.36245474]
 [-0.01958163  0.10687501  0.16590426 -3.24077582 -0.49340068]]
[False  True False  True]
[[-1.53338532 -0.77079493  2.16274676  0.89637885  0.32211041]
 [-0.48750183  2.67493798  1.50098849 -0.06362239  1.15835726]]
[ True False  True  True]
[[ 1.63691799 -0.53269813  0.01980702  0.13260588 -2.36245474]
 [-0.01958163  0.10687501  0.16590426 -3.24077582 -0.49340068]
 [-0.48750183  2.67493798  1.50098849 -0.06362239  1.15835726]]
[[1.63691799 0.          0.01980702 0.13260588 0.          ]
 [0.          0.          2.16274676 0.89637885 0.32211041]
 [0.          0.10687501 0.16590426 0.          0.          ]
 [0.          2.67493798 1.50098849 0.          1.15835726]]
```

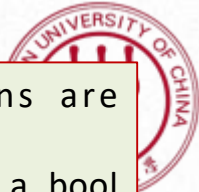
numpy

- 课堂思考与练习

- 请计算下列数组data中姓氏为Smith的同学的成绩的平均值和标准差

```
data = np.array([[ 'Bob', 'Smith', '12', '2019-02-01'],  
                 [ 'Joe', 'Smith', '13', '2018-03-07'],  
                 [ 'Roy', 'Ratner', '12', '2019-02-05'],  
                 [ 'Rita', 'Smith', '14', '2017-02-16'],  
                 [ 'Alice', 'Holmes', '11', '2020-02-29'],  
                 [ 'Wool', 'Smith', '15', '2016-02-14']])
```





numpy

• Numpy实例

– 矩阵的一些基本运算 $a = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, b = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

```

a = np.ones((2,2))
b = np.eye(2)

print (a > 2) # 判断各个元素是否大于2
print (a+b) # 对应元素相加
print (a-b) # 对应元素相减
print (a*b) # 对应元素相乘
print (b/a) # 对应元素相除
print ((a*2)*(b*2)) # a的各个元素先乘以2, b的各个元素先乘以2, 两个矩阵的各个元素再相乘

```

```

[[False False]
 [False False]]
[[2.  1.]
 [1.  2.]]
[[0.  1.]
 [1.  0.]]
[[1.  0.]
 [0.  1.]]
[[1.  0.]
 [0.  1.]]
[[4.  0.]
 [0.  4.]]

```

- **Arithmetic** operations are element-wise
- Logical operator return a bool array
- In place operations modify the array

#也可以把矩阵和一个标量进行运算

```

print (a+ 3.3) # 对应元素相加
print (a- 3.3) # 对应元素相减
print (a* 3.3) # 对应元素相乘
print (b/ 3.3) # 对应元素相除

```

```

[[4.3  4.3]
 [4.3  4.3]]
[[-2.3 -2.3]
 [-2.3 -2.3]]
[[3.3  3.3]
 [3.3  3.3]]
[[0.3030303  0.          ]
 [0.          0.3030303]]

```


numpy

- Numpy实例

- 矩阵的一些基本运算
- 通过**向量化 (vectorization)** 的方式避免了for循环，效率高

```
In [7]: import numpy as np
```

```
In [8]: my_arr = np.arange(1000000)
```

```
In [9]: my_list = list(range(1000000))
```

Now let's multiply each sequence by 2:

```
In [10]: %time for _ in range(10): my_arr2 = my_arr * 2  
CPU times: user 20 ms, sys: 50 ms, total: 70 ms  
Wall time: 72.4 ms
```

```
In [11]: %time for _ in range(10): my_list2 = [x * 2 for x in my_list]  
CPU times: user 760 ms, sys: 290 ms, total: 1.05 s  
Wall time: 1.05 s
```

面向数组的编程，对数组元素执行批量操作
——编程思路的转变

70ms vs 1.05s

numpy

- Numpy实例

- 矩阵的一些基本运算 $a = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, b = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

```
#Logical operator
```

```
b = np. eye(2)
```

```
print( b>=1)
```

```
[[ True False]
 [False  True]]
```

- Arithmetic operations are element-wise
- **Logical** operator return a bool array
- In place operations modify the array

numpy

- Numpy实例

– 矩阵的一些基本运算 $a = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, b = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

```
#In place operations modify the array
a = np.array( [[4, 15],
               [20, 75]] )
b = np.array( [[2, 5],
               [5, 15]] )

print(a)
print(b)
a = a/b
print(a)
```

```
[[ 4 15]
 [20 75]]
[[ 2  5]
 [ 5 15]]
[[2.  3.]
 [4.  5.]]
```

a/=b出错，改成a = a/b

- Arithmetic operations are element-wise
- Logical operator return a bool array
- **In place operations** modify the array

numpy

- Math, universal functions
 - Also called ufuncs is Element-wise
- Examples:
 - np.exp
 - np.sqrt
 - np.sin
 - np.cos
 - np.isnan

```
#In place operations modify the array  
a = np.array( [[4, 15], [20, 75]] )  
print( np.sqrt(a) )
```

```
[[2.          3.87298335]  
 [4.47213595  8.66025404]]
```


numpy

- Numpy实例
 - Sum/min/max

```
import numpy as np

a = (np.array( [[2,3,4],[5,6,7]] )) # 2行3列数组, 也就是一个矩阵
print(a.sum()) # sum all entries
print(a.sum(axis=0)) # 沿着行方向, 计算每一列的和, 即[ 7  9 11]
print(a.sum(axis=1)) # sum over columns
print(a.sum(axis=1, keepdims=True)) # keepdims=True

print(a.min ())
print(a.max ())
```

```
27
[ 7  9 11]
[ 9 18]
[[ 9]
 [18]]
2
7
```

- Use the axis parameter to control which axis NumPy operates on
- Typically, the axis specified will disappear, **keepdims keeps all dimensions**

numpy

- Numpy实例
 - 矩阵拼接

```
a = np.ones((2,2)) # 2行2列矩阵, 元素都是1  
b = np.eye(2) # 2行2列单位矩阵
```

```
print (np.vstack((a,b))) # 纵向合并两个矩阵, 形成4行2列的数组  
print (np.hstack((a,b))) # 横向合并两个矩阵, 形成2行4列的数组
```

```
[[1.  1.]  
 [1.  1.]  
 [1.  0.]  
 [0.  1.]]  
[[1.  1.  1.  0.]  
 [1.  1.  0.  1.]]
```

numpy

- Numpy实例
 - Reshape变形

```
ar = np.arange(15) # 生成1维数组,
print(ar.shape) # 输出 (15L, )

print (ar.reshape(3,5)) # 1维变成2维
print(ar.reshape(3,5).shape) # 输出 (3L, 5L)
print (ar.reshape(5,3)) # 改变各维大小
print(ar.reshape(5,3).shape) # 输出 (5L, 3L)

print(ar.flatten()) # 2维变成1维
print(ar.flatten().shape) # 输出 (15L, )

(15, )
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
(3, 5)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [12 13 14]]
(5, 3)
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
(15, )
```

numpy

- Numpy实例
 - Reshape变形
 - 另一个实例

```
#reshape more  
a = np.array([1, 2, 3, 4, 5, 6])  
a = a.reshape(3, 2)  
print(a)  
a = a.reshape(2, -1)  
print(a)  
a = a.ravel() #多维数组转换为一维数组  
print(a)
```

*# Total number of elements cannot change.
Use -1 to infer axis shape
Row-major by default (MATLAB is column-major)*

```
[[1 2]  
 [3 4]  
 [5 6]]  
[[1 2 3]  
 [4 5 6]]  
[1 2 3 4 5 6]
```


numpy

- Numpy实例
 - 浅拷贝和深拷贝

```
ar = np.ones((2, 2))  
br = ar  
print(br is ar)
```

```
cr = ar.copy() # 深拷贝  
print(cr is ar)
```

True
False



numpy

- Return values
 - Numpy functions return either **views** or **copies**
 - Views share data with the original array, like references in Java/C++. Altering entries of a view, changes the same entries in the original
 - The numpy documentation says which functions return views or copies
 - `Np.copy`, `np.view` make explicit copies and views

numpy

- Return values

- Numpy functions return either **views** or **copies**.
- 与Python列表的重要区别：**数据切片**是原数据的一个**视图view**，而非拷贝copy，对于切片上的任何操作都会反映在**原数据上**
- 在实际的编程实践中，非常容易在这一点上出错
- 如果希望数据切片是原数据上的一份拷贝，应显式地调用**copy()函数**

```
import numpy as np
arr = np.arange(10)
print(arr)
arr_slice = arr[5:8] # 选取一个切片，注意该切片为原数据的视图
arr_slice[0] = 1234
print(arr) # 原数据索引为5的值发生了改变
arr_slice1 = arr[1:3].copy() # 拷贝出一个切片
arr_slice1[0] = 1234
print(arr) # 原数据索引为1的值没有发生了改变
```

```
[0 1 2 3 4 5 6 7 8 9]
[ 0      1      2      3      4 1234      6      7      8      9]
[ 0      1      2      3      4 1234      6      7      8      9]
```

numpy

- Numpy实例
 - 矩阵转置和矩阵乘法

```
ar = np.array([[1, 0], [2, 3]])  
print (ar.transpose()) # 矩阵的转置
```

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

```
print(np.dot(ar, br) ) # 矩阵乘法
```

```
[[1 2]  
 [0 3]  
 [[19 22]  
 [43 50]]
```


numpy

• Numpy实例

– 特征值、特征向量、SVD分解

```
import numpy.linalg as nplg
ar = np.array([[1, 0], [-4, 3], [1, 0, 2]])
x, y = numpy.linalg.eig(ar) # 特征值和特征向量
print("x", x)
print("y", y)
print(numpy.linalg.eig(ar)) # 特征值和特征向量
print(np.dot(ar, y[:, 0])) # 验算0号特征值和特征向量
print(x[0] * y[:, 0])
print(np.dot(ar, y[:, 1])) # 验算1号特征值和特征向量
print(x[1] * y[:, 1])
print(np.dot(ar, y[:, 2])) # 验算2号特征值和特征向量
print(x[2] * y[:, 2])
```

```
ar = np.array([[1, 0], [2, 3]])
u, s, vT = np.linalg.svd(ar) # SVD分解
print("u", u)
print("s", s)
print("vT", vT)
```

```
Sigma=np.array([ [ s[0], 0], [ 0, s[1]] ])
temp1 = np.dot(u, Sigma)
temp2 = np.dot(temp1, vT)
print(temp2)
```

```
x [2. 1. 1.]
y [[ 0.          0.40824829  0.40824829]
 [ 0.          0.81649658  0.81649658]
 [ 1.         -0.40824829 -0.40824829]]
(array([2., 1., 1.]), array([[ 0.          0.40824829  0.40824829],
 [ 0.          0.81649658  0.81649658],
 [ 1.         -0.40824829 -0.40824829]]))
[0. 0. 2.]
[0. 0. 2.]
[ 0.40824829  0.81649658 -0.40824829]
[ 0.40824829  0.81649658 -0.40824829]
[ 0.40824829  0.81649658 -0.40824829]
[ 0.40824829  0.81649658 -0.40824829]
```

```
u [[-0.16018224 -0.98708746]
 [-0.98708746  0.16018224]]
s [3.65028154 0.82185442]
vT [[-0.58471028 -0.81124219]
 [-0.81124219  0.58471028]]
[[ 1.0000000e+00 -4.4408921e-16]
 [ 2.0000000e+00  3.0000000e+00]]
```



numpy

- Numpy实例

- Broadcasting

- When operating on multiple arrays, broadcasting rules are used
 - Each dimension must match, from right-to-left
 - Dimensions of size 1 will broadcast (as if the value was repeated).
 - Otherwise, the dimension must have the same shape.
 - Extra dimensions of size 1 are added to the left as needed

```
#broadcasting
a = np.random.random((5,3))
print(a)
a = a+1
print(a)
```

```
[[0.50864461 0.82415144 0.51872029]
 [0.36315603 0.88448091 0.72948211]
 [0.76725595 0.97937764 0.62829754]
 [0.18044825 0.85757133 0.03235834]
 [0.58714485 0.17216633 0.41380195]]
[[1.50864461 1.82415144 1.51872029]
 [1.36315603 1.88448091 1.72948211]
 [1.76725595 1.97937764 1.62829754]
 [1.18044825 1.85757133 1.03235834]
 [1.58714485 1.17216633 1.41380195]]
```

5*3的矩阵，每个元素，
加上实数1

numpy

- Numpy实例
- 写入文件

```
import numpy as np
data = np.random.random((4,3))
print(data)
np.save('bin_data', data) # 存入二进制文件bin_data.npy
loaded_data = np.load('bin_data.npy')
print(data)
```

```
[[0.53469027 0.27232821 0.86685555]
 [0.31708048 0.089706    0.01579114]
 [0.56721678 0.82554299 0.10713553]
 [0.53647069 0.83037797 0.04685006]]
[[0.53469027 0.27232821 0.86685555]
 [0.31708048 0.089706    0.01579114]
 [0.56721678 0.82554299 0.10713553]
 [0.53647069 0.83037797 0.04685006]]
```

numpy

- Numpy实例
- 写入文件

```
import numpy as np
a = np.array([[3.3, -0.3], [-0.3, 5.5]])
print(a)

np.savez('data1.npz', kw=a)
data = np.load('data1.npz')
a = data['kw']
print(a)
# NPZ files can hold multiple arrays #上述实例的kw是一个名字
# np.savez_compressed similar.
```

```
[[ 3.3 -0.3]
 [-0.3  5.5]]
[[ 3.3 -0.3]
 [-0.3  5.5]]
```




numpy

- Numpy实例
- 写入文件

```
import numpy as np
ar = np.random.rand(5, 5)
print(ar)
np.savetxt('txt_data.txt', ar, fmt='%0.8f') # 存入文本文件
br=np.loadtxt('txt_data.txt', dtype=np.float32)
print(br)
```

```
[[0.03598973 0.40317196 0.06378865 0.58898706 0.75733172]
 [0.84137219 0.38056775 0.76486695 0.90368303 0.49065687]
 [0.08756266 0.83337941 0.13406773 0.07288254 0.19418188]
 [0.19500955 0.81605742 0.66445988 0.51764073 0.94448538]
 [0.54816263 0.85934666 0.23400618 0.53817725 0.14403302]]
[[0.03598973 0.40317196 0.06378865 0.58898705 0.7573317 ]
 [0.8413722  0.38056776 0.76486695 0.903683  0.49065688]
 [0.08756266 0.8333794  0.13406773 0.07288254 0.19418187]
 [0.19500954 0.81605744 0.6644599  0.5176407  0.94448537]
 [0.54816264 0.8593467  0.23400618 0.53817725 0.14403301]]
```

numpy

- Numpy实例
- 写入文件

```
import numpy as np
data = np.genfromtxt( 'cdv_data.csv', delimiter=',', names=True)
print(data)
```

```
[(1., 123., 1., 1.) (2., 124., 2., 2.) (3., 125., 3., 3.)]
```



numpy



numpy

- Load image

```
#load images  
from PIL import Image  
import numpy as np  
  
im = np.array(Image.open('./boy.png'))  
  
print(type(im))  
print(im.dtype)  
print(im.shape)
```

```
<class 'numpy.ndarray'>  
uint8  
(543, 520, 3)
```


numpy

- show image

```
import numpy as np
import matplotlib.pyplot as plt
def plot_image(im, h=8, **kwargs):
    """
    Helper function to plot an image.
    """
    y = im.shape[0]
    x = im.shape[1]
    w = (y/x) * h
    plt.figure(figsize=(w,h))
    plt.imshow(im, interpolation="none", **kwargs)
    plt.axis('off')

plot_image(im)
```



更多资料请见<https://note.nkmk.me/en/python-numpy-image-processing/>
<http://www.degeneratestate.org/posts/2016/Oct/23/image-processing-with-numpy/>

numpy

- Image arrays
- Images are 3D arrays: width, height, and channels
- Common image formats:
 - height x width x RGB (band-interleaved)
 - height x width (band-sequential)
- Gotchas:
 - Channels may also be BGR (OpenCV does this)
 - May be [width x height], not [height x width]



numpy

- Make image

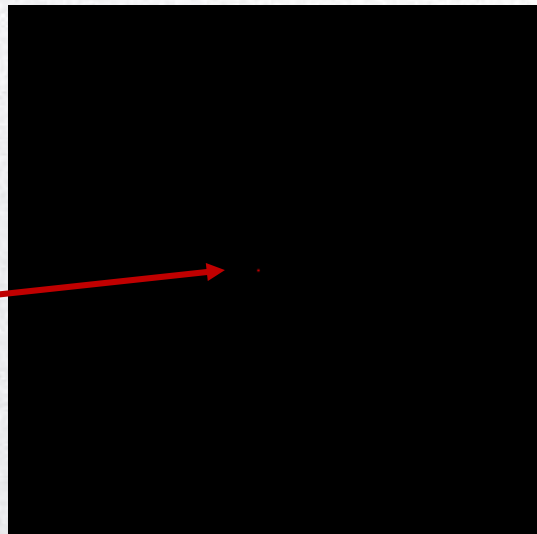
```
#make image
from PIL import Image
import numpy as np

w, h = 512, 512
data = np.zeros((h, w, 3), dtype=np.uint8)
data[256, 256] = [255, 0, 0]
data[255, 256] = [255, 0, 0]
data[256, 255] = [255, 0, 0]
data[255, 255] = [255, 0, 0]
img = Image.fromarray(data, 'RGB')
img.save('my.png')


#img.show()

im = np.array(Image.open('./my.png'))
plot_image(im)
```

红点



numpy

- Math, upcasting
- Just as in Python and Java, the result of a math operator is cast to the **more general or precise datatype**
 - `uint64 + uint16 → uint64`
 - `float32 / int32 → float32`
- Warning: upcasting does not prevent overflow/underflow. You must manually cast first.

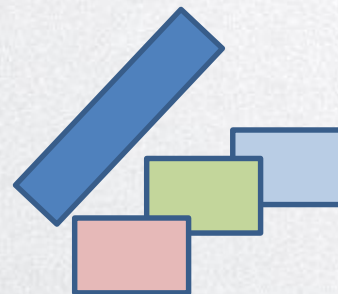
- Use case: images often stored as `uint8`
- You should convert to `float32` or `float64` before doing math

numpy

- Broadcasting

- 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.

此内容，不做
强制要求





numpy

- 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)

此内容，不做
强制要求



numpy



numpy

- 总结: Tips to avoid bugs
 - Know what your **datatypes** are
 - Check whether you have a **view** or a **copy**
 - Use **matplotlib** for sanity checks
 - Use **pdb**(Python 调试器) or print to check each step of your computation
 - Know **np.dot** vs **np.mult**

Python 调试器之pdb

使用PDB的方式有两种:

1. 单步执行代码,通过命令 `python -m pdb xxx.py` 启动脚本, 进入单步执行模式

pdb命令行:

1) 进入命令行Debug模式, `python -m pdb xxx.py`

numpy

- 参考资料



部分内容来自

<https://www.cs.cornell.edu/courses/cs4670/2018sp/lec08-numpy.pptx>