3. Math primer and preface to Deep Learning

3.2 - Linear Algebra Operations in Python

Rodolfo Sabino

Julho, 2020

Departamento de de Computação

Index

Introduction To The Numpy library

Introduction To The Numpy library

Numpy provides support for large multidimensional arrays and matrices

It provides a set of high level mathematical functions.

Python Library Installation

Python Library Installation

Pip is the package manager for python libraries.

Pip is included by default starting from the Python 3.4 installer.

Install numpy by running the following command:

```
pip3 install numpy
```

API Reference

API Reference

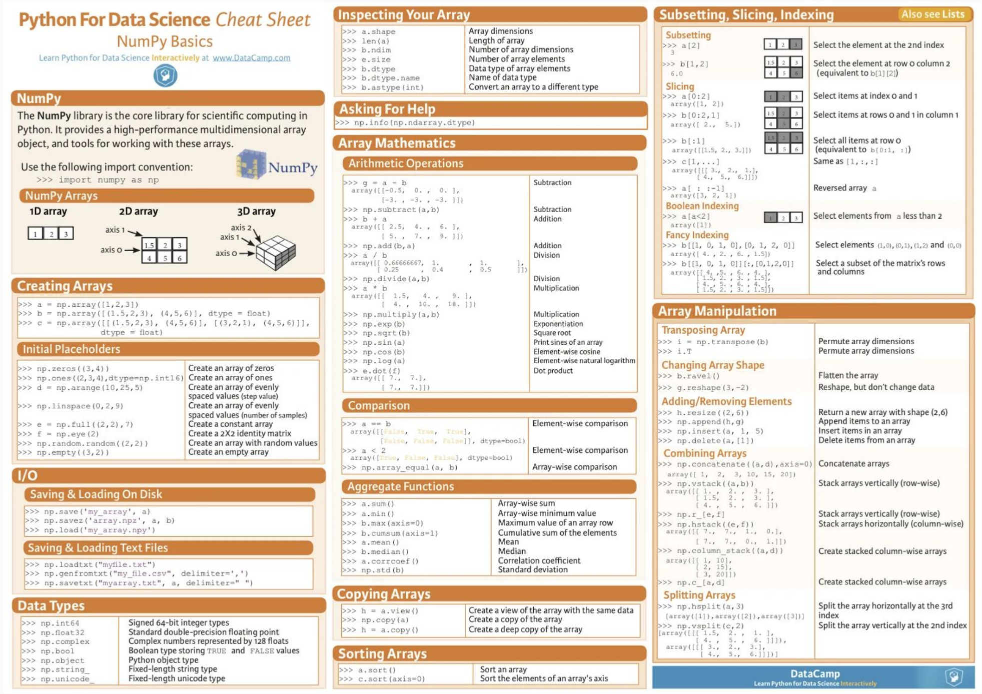
The numpy website provides a comprehensive list with functions with their use.

https://numpy.org/devdocs/reference/index.html

API Reference

Cheat sheets help by providing an overview of the common functionalities of the library.

Source link



Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com

NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays

Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

`>>> np.zeros((3,4))`	Create an array of zeros
`>>> np.ones((2,3,4),dtype=np.int16)`	Create an array of ones
`>>> d = np.arange(10,25,5)`	Create an array of evenly spaced values (step value)
`>>> np.linspace(0,2,9)`	Create an array of evenly spaced values (number of samples)
`>>> e = np.full((2,2),7)`	Create a constant array
`>>> f = np.eye(2)`	Create a 2X2 identity matrix
`>>> np.random.random((2,2))`	Create an array with random values
`>>> np.empty((3,2))`	Create an empty array

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

`>>> np.int64`	Signed 64-bit integer types
`>>> np.float32`	Standard double-precision floating point
`>>> np.complex`	Complex numbers represented by 128 floats
`>>> np.bool`	Boolean type storing TRUE and FALSE values
`>>> np.object`	Python object type
`>>> np.string_`	Fixed-length string type
`>>> np.unicode_`	Fixed-length unicode type

Inspecting Your Array

`>>> a.shape`	Array dimensions
`>>> len(a)`	Length of array
`>>> b.ndim`	Number of array dimensions
`>>> e.size`	Number of array elements
`>>> b.dtype`	Data type of array elements
`>>> b.dtype.name`	Name of data type
`>>> b.astype(int)`	Convert an array to a different type

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

`>>> g = a - b`	Subtraction
`array([[-0.5, 0. , 0. ],`	
`[-3. , -3. , -3. ]])`	
`>>> np.subtract(a,b)`	Subtraction
`>>> b + a`	Addition
`array([[ 2.5, 4. , 6. ],`	
`[ 5. , 7. , 9. ]])`	
`>>> np.add(b,a)`	Addition
`>>> a / b`	Division
`array([[ 0.66666667, 1. , 1. ],`	
`[ 0.25 , 0.4 , 0.5 ]])`	
`>>> np.divide(a,b)`	Division
`>>> a * b`	Multiplication
`array([[ 1.5, 4. , 9. ],`	
`[ 4. , 10. , 18. ]])`	
`>>> np.multiply(a,b)`	Multiplication
`>>> np.exp(b)`	Exponentiation
`>>> np.sqrt(b)`	Square root
`>>> np.sin(a)`	Print sines of an array
`>>> np.cos(b)`	Element-wise cosine
`>>> np.log(a)`	Element-wise natural logarithm
`>>> e.dot(f)`	Dot product
`array([[ 7., 7.],`	
`[ 7., 7.]])`	

Comparison

`>>> a == b`	Element-wise comparison
`array([[False, True, True],`	
`[False, False, False]], dtype=bool)`	
`>>> a < 2`	Element-wise comparison
`array([True, False, False], dtype=bool)`	
`>>> np.array_equal(a, b)`	Array-wise comparison

Aggregate Functions

`>>> a.sum()`	Array-wise sum
`>>> a.min()`	Array-wise minimum value
`>>> b.max(axis=0)`	Maximum value of an array row
`>>> b.cumsum(axis=1)`	Cumulative sum of the elements
`>>> a.mean()`	Mean
`>>> b.median()`	Median
`>>> a.corrcoef()`	Correlation coefficient
`>>> np.std(b)`	Standard deviation

Copying Arrays

`>>> h = a.view()`	Create a view of the array with the same data
`>>> np.copy(a)`	Create a copy of the array
`>>> h = a.copy()`	Create a deep copy of the array

Sorting Arrays

`>>> a.sort()`	Sort an array
`>>> c.sort(axis=0)`	Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Also see Lists

Subsetting	
`>>> a[2]`	Select the element at the 2nd index
`3`	
`>>> b[1,2]`	Select the element at row 0 column 2 (equivalent to `b[1][2]`)
`6.0`	
Slicing	
`>>> a[0:2]`	Select items at index 0 and 1
`array([1, 2])`	
`>>> b[0:2,1]`	Select items at rows 0 and 1 in column 1
`array([ 2., 5.])`	
`>>> b[:1]`	Select all items at row 0 (equivalent to `b[0:1, :]`)
`array([[1.5, 2., 3.]])`	
`>>> c[1,...]`	Same as `[1,:,:]`
`array([[[ 3., 2., 1.],`	
`[ 4., 5., 6.]]])`	
`>>> a[ : :-1]`	Reversed array `a`
`array([3, 2, 1])`	
Boolean Indexing	
`>>> a[a<2]`	Select elements from `a` less than 2
`array([1])`	
Fancy Indexing	
`>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]`	Select elements (1,0),(0,1),(1,2) and (0,0)
`array([ 4. , 2. , 6. , 1.5])`	
`>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]`	Select a subset of the matrix's rows and columns
`array([[ 4. ,5. , 6. , 4. ],`	
`[ 1.5, 2. , 3. , 1.5],`	
`[ 4. , 5. , 6. , 4. ],`	
`[ 1.5, 2. , 3. , 1.5]])`	

Array Manipulation

Transposing Array	
`>>> i = np.transpose(b)`	Permute array dimensions
`>>> i.T`	Permute array dimensions
Changing Array Shape	
`>>> b.ravel()`	Flatten the array
`>>> g.reshape(3,-2)`	Reshape, but don't change data
Adding/Removing Elements	
`>>> h.resize((2,6))`	Return a new array with shape (2,6)
`>>> np.append(h,g)`	Append items to an array
`>>> np.insert(a, 1, 5)`	Insert items in an array
`>>> np.delete(a,[1])`	Delete items from an array
Combining Arrays	
`>>> np.concatenate((a,d),axis=0)`	Concatenate arrays
`array([ 1, 2, 3, 10, 15, 20])`	
`>>> np.vstack((a,b))`	Stack arrays vertically (row-wise)
`array([[ 1. , 2. , 3. ],`	
`[ 1.5, 2. , 3. ],`	
`[ 4. , 5. , 6. ]])`	
`>>> np.r_[e,f]`	Stack arrays vertically (row-wise)
`>>> np.hstack((e,f))`	Stack arrays horizontally (column-wise)
`array([[ 7., 7., 1., 0.],`	
`[ 7., 7., 0., 1.]])`	
`>>> np.column_stack((a,d))`	Create stacked column-wise arrays
`array([[ 1, 10],`	
`[ 2, 15],`	
`[ 3, 20]])`	
`>>> np.c_[a,d]`	Create stacked column-wise arrays
Splitting Arrays	
`>>> np.hsplit(a,3)`	Split the array horizontally at the 3rd index
`[array([1]),array([2]),array([3])]`	
`>>> np.vsplit(c,2)`	Split the array vertically at the 2nd index
`[array([[[ 1.5, 2. , 1. ],`	
`[ 4. , 5. , 6. ]]]),`	
`array([[[ 3., 2., 3.],`	
`[ 4., 5., 6.]]])]`	

DataCamp
Learn Python for Data Science Interactively

Data Types

Data Types

Some of the data types we expect find when working with numpy are:

- `bool`
- `int_`
- `float_`
- `complex_`

Data Types

The following variables contain values of each data type:

- `b = True`
- `i = 10`
- `f = 5.0`
- `c = 1+2j`

Data Types

Creating vector can be done by the following line. This function creates a vector of type `np.int_`.

```
v = np.array([[1,2,3]])
```

A a specific data type can be explicitly defined during creation like the following example. This function creates a vector of type np.float_.

```
v = np.array([[1,2,3]],dtype=np.float_)
```

Data Types

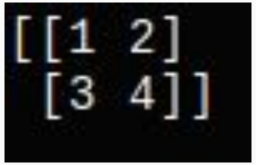
A multidimensional matrix can be created as follows:

```
M = np.array([[1,2],[3,4]])
```

Variables can be printed to the standard output using the `print` function:

```
print(M)
```

Output:



```
[[1 2]
 [3 4]]
```

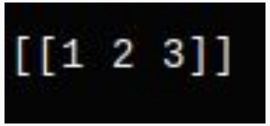
Data Types

The Identity matrix (square matrix with 1s in the diagonal and 0s everywhere else) can be created by the function np.eye(N) where N is the square matrix dimension.

Data Types

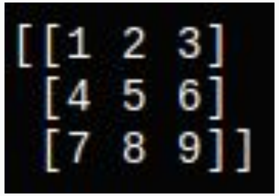
Numpy matrices are stored in row-major order. Meaning that each consecutive values in matrices describe a row to the next.

- `print(mp.array([1,2,3]))`



```
[[1 2 3]]
```

- `print(np.array([[1,2,3],[4,5,6],[7,8,9]]))`



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

Operators and Functions

Operators and Functions

Indexing elements of a vector or matrix can be done as follows:

Given a 2D matrix **M**:

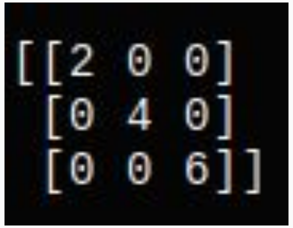
- `M[2]` Selects the third row;
- `M[1,2]` Selects the second row, third column;
- `M[:]` Selects all elements;
- `M[:,0]` Selects the first column;
- `M[:,:2]` Selects the first two columns;
- `M[:,1:3]` Selects the second to third columns.

Operators and Functions

Operators + – * / are used for itemwise operations.

```
M = np.array([[2,0,0],[0,2,0],[0,0,2]])
v = np.array([[1,2,3]])
print(M*v.T)
```

Output:

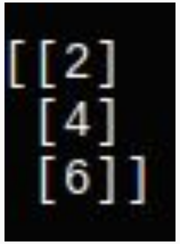


```
[[2 0 0]
 [0 4 0]
 [0 0 6]]
```

Operators and Functions

The operator @ is used for matrix multiplication. Alternatively the function `np.matmul()` can also be used.

```
M = np.array([[2,0,0],[0,2,0],[0,0,2]])
v = np.array([[1,2,3]])
print(M@v.T)
```



```
[[2]
 [4]
 [6]]
```

Operators and Functions

The `mp.transpose()` function is used for matrix transpose. M.T can also be used as a shorthand for the same objective.

- `M_transpose1 = np.transpose(M)`
- `M_transpose2 = M.T`

Operators and Functions

The dot product between two vectors can be achieved by @ or by the `np.vdot()` function.

Let v be a row vector:

- `v@v.T`
- `np.vdot(v,v.T)`

Operators and Functions

The cross product between two 3D vectors can be computed by the following function.

Let x and y be two non zero, non collinear vectors in R^3. z Is a vector that is orthogonal to both x and y.

```
z = np.cross(x,y)
```

Operators and Functions

More linear algebra functions, such as for computing the determinant, matrix inverse, vector norm and eigenvectors can be found under the `np.linalg.*` namespace.

https://numpy.org/doc/stable/reference/routines.linalg.html

Capítulo 1

Chairs vary in design. An armchair has armrests fixed to the seat; a recliner is upholstered and under its seat is a mechanism that allows one to lower the chair's back and raise into place a fold-out footrest; a rocking chair has legs fixed to two long curved slats; a wheelchair has wheels fixed to an axis under the seat.

Quem descobriu o Brasil?

Chapter 1

Chair comes from the early 13th-century English word chaere, from Old French chaiere ("chair, seat, throne"), from Latin cathedra ("seat").

Armchair	Recliner
Has armrests.	Has two long curved slats.
Recliner	**Wheelchair**
Allow one to lower the back.	Has wheels.