2-Numpy

March 3, 2021

1 Numpy

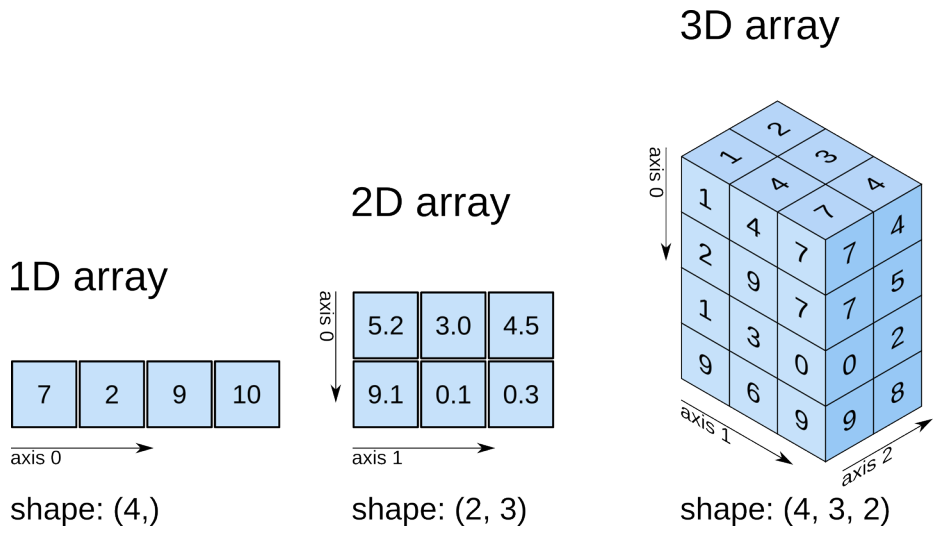
Numpy è una libreria creata per permettere la creazione di ***array n-dimensionali in python***, le caratteristiche principali di numpy è che questa libreria è **estremamente veloce, versatile e usata in molte altre librerie**, come infatti vedremo in seguito altre librerie sono basate su di essa per permettere di fare operazioni più veloci e avere controllo sulla memoria.

```
Il motivo per cui numpy è così veloce è dovuto al fatto che è basata su linguaggi a più basso li
```

```
Numpy dovrebbe essere già installato con anaconda, qualora però non lo fosse installatelo seguen
```

1.1 Cosa sono gli array ?

Gli array sono un concetto informatica ispirato dalla nozione di vettori, matrici o più in generale tensori usati in geometria e matematica. Il concetto di array è già stato introdotto indirettamente nel capitolo 1 con la definzione di strutture di dati in particolare con la definizione di liste e dizionari, per avere una visione su cosa siano questi array usiamo un immagine di numpy.



Come possiamo notare dall'immagine un array 1d è equivalemente a una lista in python, il problema sorge qualora ci spostiamo in array 2d o superiori per cui non abbiamo un'istruzione diretta per la creazione di tali elementi, in tal caso facciamo ricorso alla libreria notando che numpy creerà una lista di liste legata al numero di dimensioni considerata. Vediamo però il suo utilizzo attraverso la libreria.ù

1.2 Creazione di array e operazioni

1.2.1 Creazione di array

Per creare un array in numpy in primo passo è necessario importare la libreria dopodiché è possibile inizializzare un array decidendo se dovrà contenere zeri o uni, attenzione i valori che potreste vedere potrebbero essere valori approssimata seconda la macchina essere vicini a quelli.

[1]:
```
import numpy as np
#notare bene le doppie paretesi
ones_1d = np.ones((4,)) #contiene solo 4 elementi
zeros_1d = np.zeros((4,))
print('Array 1dimensionale contenenti uni', ones_1d)
print('Array 1dimensionale contenenti zeri', zeros_1d)
```

```
Array 1dimensionale contenenti uni [1. 1. 1. 1.]
Array 1dimensionale contenenti zeri [0. 0. 0. 0.]
```

Qualora volessi creare array 2D l'unica modifica che è necessaria è aggiungere il numero della seconda dimensione all'interno della doppia parentesi.

[2]:
```
ones_2d = np.ones((2,3))
zeros_2d = np.zeros((2,3))
print('Array 2dimensioni contenenti uni:\n', ones_2d)
print('Array 2dimensioni contenenti zeri:\n', zeros_2d)
```

```
Array 2dimensioni contenenti uni:
 [[1. 1. 1.]
 [1. 1. 1.]]
Array 2dimensioni contenenti zeri:
 [[0. 0. 0.]
 [0. 0. 0.]]
```

Come è possibile notare l'array 1d è l'analogo della lista, mentre quello 2d è una lista di liste questo poiché in python come abbiamo detto non esistono altre strutture oltre a questa per la memorizzazione di dati in maniera ordinata. Per quanto riguarda gli array 3d ripetiamo i procedimenti precedenti.

[3]:
```
ones_3d = np.ones((4,3,2))
zeros_3d = np.zeros((4,3,2))
print('Array 3dimensioni contenenti uni:\n', ones_3d)
print('Array 3dimensioni contenenti zeri:\n', zeros_3d)
```

```
Array 3dimensioni contenenti uni:
 [[[1. 1.]
  [1. 1.]
  [1. 1.]]

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

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

 [[1. 1.]
  [1. 1.]
  [1. 1.]]]
Array 3dimensioni contenenti zeri:
 [[[0. 0.]
  [0. 0.]
  [0. 0.]]

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

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

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

1.2.2 Operazioni su array

Su questi array è possibile effettuare delle operazioni come se fossero delle variabili e molte altre. #### Addizione #### Per l'operazione di somma tra due array a noi basta usare + oppure `np.add`.

[4]:
```
print(ones_1d+ones_1d)
```

```
[2. 2. 2. 2.]
```

[5]:
```
print(np.add(ones_1d, ones_1d))
```

```
[2. 2. 2. 2.]
```

Sottrazione Per l'operazione di differenza tra due array a noi basta usare - oppure `np.substract`.

[6]:
```
print(ones_2d - ones_2d)
```

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

Moltiplicazione Per l'operazione di moltiplicazione tra due array a noi basta usare * oppure `np.multiply`.

[7]:
```
print(ones_2d * zeros_2d)
```

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

Divisione Per l'operazione di divisione tra due array a noi basta usare / oppure `np.divide`.

[8]:
```
print(ones_2d / 2)
```

```
[[0.5 0.5 0.5]
 [0.5 0.5 0.5]]
```

```
Notare bene che l'ultima operazione che ho fatto non è tra due array, il motivo per cui funziona
```

Ci sono molte altre operazioni nella librerie che sarebbero troppo lunghe da elencare qui, per avere un quadro generale di tutto ciò potete consultare questo **cheatsheet**, mentre se aveste dubbi potete consultare la **documentazione**.

1.3 Accesso ad elementi negli array

Per accedere agli elementi degli si applicano le stesse regole della lista, è necessario quindi ricordare che gli indici partono da 0, per quanto riguardo l'accesso dei singoli elementi in array multidimensionali è necessario usare un numero di indici uguale alla dimensione dell'array, mentre nel caso volessimo accedere a elementi multipli basta definire un intervallo come nelle liste, vediamo degli esempi per chiarire ciò.

[9]:
```
#selezionare uno o più elementi in array 1d
print('Array 1d')
print('Elemento al 2 indice', ones_1d[2])
print('Elementi dal primo al penultimo', ones_1d[:-2])
print('Tutti gli elementi', ones_1d[:])
#selezionare uno o più elementi in 2d
print('Array 2d')
print('elemento in posizione (1,2)', ones_2d[1,2])
print('Prima riga', ones_2d[0,:])
print('Prima colonna', ones_2d[:,0])
```

```
Array 1d
Elemento al 2 indice 1.0
Elementi dal primo al penultimo [1. 1.]
Tutti gli elementi [1. 1. 1. 1.]
Array 2d
elemento in posizione (1,2) 1.0
Prima riga [1. 1. 1.]
Prima colonna [1. 1.]
```

1.4 Tipi di dati in numpy

Numpy essendo basato su linguaggi a più basso livello fornisce maggiore informazioni su come sono costruiti i dati, vediamolo guardando le informazioni su un array.

[10]:
```
np.info(ones_2d)
```

```
class:  ndarray
shape:  (2, 3)
strides:  (24, 8)
itemsize:  8
aligned:  True
contiguous:  True
fortran:  False
data pointer: 0x2c7924e8180
byteorder:  little
byteswap:  False
type: float64
```

Come possiamo vedere abbiamo informazioni sull'array: - la classe - shape: la forma dell'array - strides: quanti bytes sono distanti da una cella di memoria all'altra - itemsize: la sua lunghezza in byte, - aligend: se è allineata in memoria - contiguous: ci dice se l'array è costruito usando la notazione in C - fortran è l'analogo di contiguous con la condizione posta sul linguaggio Fortran - il data pointer definisce l'indirizzo di memoria dove sono contenuti i dati - byteorder definisce quale sia l'ordine di questo tipo di dato- - byteswap invece ci dice se vogliamo scambiare il byteorder. - type: il tipo di dato con il numero di bit

Il punto centrale risulta però essere il type che ci dice che è di tipo float64 ovvero di tipo float come già visto in python, con l'aggiunta del fatto che ogni numero viene memorizzato in 64bit. Il 64bit viene considerato il formato standard poiché buon compresso tra velocità e precisione, qualora però sia necessaria più velocità e minore precisione potete convertire l'array in un formato con un numero di bit minore usando il comando `.astype('nuovo_tipo')`dove il nuovo tipo deve essere tra quelli ammessi, guardare questa **link** per maggiori info.

[11]:
```
#creo un nuovo array basandomi sul precedente con il tipo float32
ones_2d_32bit = ones_2d.astype('float32')
np.info(ones_2d_32bit)
```

```
class:  ndarray
shape:  (2, 3)
strides:  (12, 4)
itemsize:  4
aligned:  True
contiguous:  True
fortran:  False
data pointer: 0x2c79280ea50
byteorder:  little
byteswap:  False
type: float32
```

State attenti ad usare un numero di bit basso per numeri particolarmente grandi, poiché potreste

---

COMPLIMENTI AVETE COMPLETATO LA LEZIONE DI NUMPY!