27. Concatenating, Flattening và thêm Dimensions

Flattening

Trong phần này tôi xin giới thiệu cách chuyển một mảng nhiều chiều về mảng chỉ có môt chiều mà tôi gọi đó là làm phẳng (flatten) mảng hoặc mảng có shape mong muốn khác. Thư viện numpy cung cấp ta hai phương thức giúp ta làm việc này dễ dàng: flatten() và ravel().

Phưng thức flatten() sử dụng một tham số từ khoá tùy chọn "order". Ý nghĩa của “order” được miêu tả dưới đây. Giá trị mặc định là "C".

Ý nghĩa “order” cho cả flatten() và ravel(): với mảng a[i1][i2][i3]…[in]
'C': trật tự C-like , với chỉ số trục cuối cùng (in) thay đổi nhanh nhất, quay trở lại chỉ số trục đầu tiên thay đổi chậm nhất (i1). "C" là mặc định!

'F': trật tự Fortran-like, với chỉ số đầu tiên (i1) thay đổi nhanh nhất, và chỉ số cuối cùng thay đổi chậm nhất (in).

'A': trật tự sẽ là Fortran-like nếu mảng "a" là “Fortran *contiguous*” trong bộ nhớ, tương tự trật tự sẽ là c-like nếu ngược lại.

'K': đọc các phần tử theo thứ tự chúng xuất hiện trong bộ nhớ, ngoại trừ việc đảo chiều dữ liệu khi chỉ số là âm.

Ví dụ:

```
import numpy as np
np.random.seed(1234)
A = np.random.randint(100,size=(3,4,2))
print "A = ", A
Flattened_X = A.flatten()
print "Flattened_X = ", Flattened_X
print "A.flatten(order='C') = ", A.flatten(order="C")
print "A.flatten(order='F') = ", A.flatten(order="F")
print "A.flatten(order='A') = ", A.flatten(order="A")



Output:



A =  [[[47 83]

  [38 53]

  [76 24]

  [15 49]]



 [[23 26]

  [30 43]

  [30 26]

  [58 92]]



 [[69 80]

  [73 47]

  [50 76]

  [37 34]]]

Flattened_X =  [47 83 38 53 76 24 15 49 23 26 30 43 30 26 58 92 69 80 73 47 50 76 37 34]

A.flatten(order='C') =  [47 83 38 53 76 24 15 49 23 26 30 43 30 26 58 92 69 80 73 47 50 76 37 34]

A.flatten(order='F') =  [47 23 69 38 30 73 76 30 50 15 58 37 83 26 80 53 43 47 24 26 76 49 92 34]

A.flatten(order='A') =  [47 83 38 53 76 24 15 49 23 26 30 43 30 26 58 92 69 80 73 47 50 76 37 34]
```

Sử dụng ravel(), thứ tự của các phần tử trong mảng trả về bởi ravel () thường là kiểu "C-style".

Cú pháp: ravel (a, order = 'C')

Ravel trả về một mảng một chiều. Bản sao được thực hiện chỉ khi cần thiết.

Thông số từ khóa tùy chọn "order" có thể là 'C', 'F', 'A', hoặc 'K'
Ví dụ:

```
import numpy as np
np.random.seed(1234)
A = np.random.randint(100,size=(3,4,2))
print "A = ", A
RavelX = A.ravel()
print "RavelX = ", RavelX
print "A.ravel(order='C') = ", A.ravel(order="C")
print "A.ravel(order='F') = ", A.ravel(order="F")
print "A.ravel(order='A') = ", A.ravel(order="A")



Output:



A =  [[[47 83]

  [38 53]
```

```
 [76 24]
 [15 49]]

[[23 26]
 [30 43]
 [30 26]
 [58 92]]

[[69 80]
 [73 47]
 [50 76]
 [37 34]]]
RavelX =  [47 83 38 53 76 24 15 49 23 26 30 43 30 26 58 92 69 80 73 47 50 76 37 34]
A.ravel(order='C') =  [47 83 38 53 76 24 15 49 23 26 30 43 30 26 58 92 69 80 73 47 50 76 37 34]
A.ravel(order='F') =  [47 23 69 38 30 73 76 30 50 15 58 37 83 26 80 53 43 47 24 26 76 49 92 34]
A.ravel(order='A') =  [47 83 38 53 76 24 15 49 23 26 30 43 30 26 58 92 69 80 73 47 50 76 37 34]
```

Qua 2 ví dụ trên có thể thấy cả hai phương thức ravel() hay flatten() đều trả về list kết quả như nhau. Vậy tại sao cần sinh ra 2 phương thức để thực hiện cùng một công việc ? Hãy cùng theo dõi ví dụ bên dưới:

```
>>> import numpy as np
# Khoi tao 2 mang A, B voi cung seed -> Ta se nhan duoc 2 mang giong nhau
>>> np.random.seed(10)
>>> A=np.random.randint(20,size=(2,3))
>>> np.random.seed(10)
>>> B=np.random.randint(20,size=(2,3))
>>> A
array([[ 9,  4, 15],
       [ 0, 17, 16]])
>>> B
array([[ 9,  4, 15],
       [ 0, 17, 16]])
# Lam phang 2 mang A,B voi phuong thuc flatten() va Ravel()
>>> FlattenA = A.flatten()
>>> RavelB = B.ravel()
>>> FlattenA
array([ 9,  4, 15,  0, 17, 16])
>>> RavelB
array([ 9,  4, 15,  0, 17, 16])
# Thay đổi mảng được trả về bởi 2 phương thức
>>> FlattenA[0] = 1
>>> RavelB[0]= 1
# Thay đổi mảng được trả về bởi flatten() -> mảng gốc không thay đổi
>>> A
array([[ 9,  4, 15],
       [ 0, 17, 16]])
# Thay đổi mảng được trả về bởi ravel() -> mảng gốc thay đổi theo
>>> B
array([[ 1,  4, 15],
       [ 0, 17, 16]])
>>>
```

Như vậy flatten() luôn trả về một copy của mảng gốc trong khi ravel() là trả về một view của mảng gốc. Dẫn đến kết quả là nếu ta thay đổi giá trị mảng được trả về bởi ravel() thì mảng gốc cũng sẽ thay đổi theo. Bù lại thì ravel sẽ thường xử lý nhanh hơn bởi không cần memory để thực hiện copy. Nhưng các bạn cần cẩn thận với việc chỉnh sửa mảng được trả về bởi ravel().

Reshape

Làm phẳng một mảng chỉ là một cách để thay đổi chiều của mảng. Một mảng có thể thay đổi thành bất kỳ chiều nào miễn là không thay đổi số lượng phần

tử trong mảng. Việc này có thể thực hiện qua hàm reshape(). Phương thức reshape() đã được chúng tôi nhắc đến trong “bài 24: numpy slicing indexing”. Trong phần này tôi sẽ nêu chi tiết hơn về ý nghĩa cũng như cú pháp của nó.

Reshape() trả về mảng mới có shape mới mà không thay đổi dữ liệu của nó.

Cú pháp là **reshape(a, newshape, order = 'C'),** ý nghĩa của thông số: a là mảng cần reshape, newshape là một tuple sẽ nêu rõ kích thước các chiều muốn chuyển về, “order” sẽ nhận một trong các giá trị sau 'C', 'F', 'A' với ý nghĩa giống trong flatten() hay ravel().

Quan sát ví dụ sau để rõ hơn cách sử dụng reshape().

```
import numpy as np
X = np.array(range(50))
print "X = ", X
print "changing to (10,5)"
Y = X.reshape((10,5))
print "Y = ", Y
print "changing to (2,5,5)"
Y = X.reshape((2,5,5))
print "Y = ",Y


Output:


X =  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49]

changing to (10,5)

Y =  [[ 0  1  2  3  4]

 [ 5  6  7  8  9]

 [10 11 12 13 14]

 [15 16 17 18 19]

 [20 21 22 23 24]

 [25 26 27 28 29]

 [30 31 32 33 34]

 [35 36 37 38 39]

 [40 41 42 43 44]

 [45 46 47 48 49]]

changing to (2,5,5)

Y =  [[[ 0  1  2  3  4]

  [ 5  6  7  8  9]

  [10 11 12 13 14]

  [15 16 17 18 19]

  [20 21 22 23 24]]


 [[25 26 27 28 29]

  [30 31 32 33 34]

  [35 36 37 38 39]

  [40 41 42 43 44]

  [45 46 47 48 49]]]
```

Ghép các mảng (Concatenating Arrays)

Trong quá trình xử lý dữ liệu, rất nhiều bài toán đòi hỏi cần ghép nối nhiều mảng với nhau để mở rộng hàng hay cột thì khĩ thuật “Concatenating Arrays” tỏ ra rất hiệu quả.

Cú pháp: numpy.concatenate((a1,a2,…), axis=0)

Trong đó: (a1,a2,…) là list các mảng cần ghép.

Trong ví dụ sau, chúng ta nối ba mảng một chiều vào một mảng qua phương thức concatenate()

```
import numpy as np
x = np.array([110,202])
y = np.array([108,70,6])
z = np.array([10,3,5])
c = np.concatenate((x,y,z))
print "x = ",x
print "y = ", y
print "c = ", c


Output:


x =  [110 202]

y =  [108  70   6]
```

```
c =  [110 202 108  70   6  10   3   5]
```

Khi ghép các mảng đa chiều, chúng ta có thể ghép các mảng theo trục. Điều kiện là mảng phải có cùng một size tương ứng với trục muốn ghép khi dùng concatenate(). Giá trị mặc định là axis = 0: Dưới đây là ví dụ cho cả hai trường hợp axis=0 và axis = 1, các bạn chú ý kết quả để có thể hiểu được cách ghép.

```
import numpy as np
np.random.seed(1234)
x =np.random.randint(100,size=(2,2,3))
y =np.random.randint(100,size=(2,2,3))
print "x = ",x
print "y = ",y
z = np.concatenate((x,y))
print "np.concatenate((x,y)) = ",z
print "change axis=1"
z = np.concatenate((x,y),axis = 1)
print "np.concatenate((x,y),axis = 1) = ",z


Output:

x =  [[[47 83 38]

  [53 76 24]]



 [[15 49 23]

  [26 30 43]]]

y =  [[[30 26 58]

  [92 69 80]]



 [[73 47 50]

  [76 37 34]]]

np.concatenate((x,y)) =  [[[47 83 38]

  [53 76 24]]



 [[15 49 23]

  [26 30 43]]



 [[30 26 58]

  [92 69 80]]



 [[73 47 50]

  [76 37 34]]]

change axis=1

np.concatenate((x,y),axis = 1) =  [[[47 83 38]

  [53 76 24]

  [30 26 58]

  [92 69 80]]



 [[15 49 23]

  [26 30 43]

  [73 47 50]

  [76 37 34]]]
```

Adding New Dimensions

Các tham số mới có thể được thêm vào mảng bằng cách sử dụng kết hợp slicing và np.newaxis. Chúng tôi minh họa kỹ thuật này bằng một ví dụ:

```
import numpy as np
np.random.seed(1234)
x = np.random.randint(100,size=10)
print "x = ",x
print "x[:,np.newaxis] = " , x[:,np.newaxis]
print "x[len(x)/2:,np.newaxis] = ", x[len(x)/2:,np.newaxis]

Output:

x =  [47 83 38 53 76 24 15 49 23 26]

x[:,np.newaxis] =  [[47]

 [83]

 [38]
```

```
 [53]

 [76]

 [24]

 [15]

 [49]

 [23]

 [26]]

x[len(x)/2:,np.newaxis] =  [[24]

 [15]

 [49]

 [23]

 [26]]
```

Xếp chồng các mảng:

Ta có thể xếp chồng các mảng theo hàng hoặc theo cột qua hai phương thức sau: row_stack() và column_stack(). Bạn có thể xếp chồng nhiều vectors tùy ý

```
import numpy as np
x = np.array(range(4))
np.random.seed(1234)
y = np.random.randint(100,size=4)
print "x = ", x
print "y = ", y
print "np.row_stack((x,y)) = ", np.row_stack((x,y))
print "np.column_stack((x,y)) = ", np.column_stack((x,y))
y = np.random.randint(100,size=(4,3))
print "y = ", y
print "np.column_stack((x,y)) = ", np.column_stack((x,y))


Output:


x =  [0 1 2 3]

y =  [47 83 38 53]

np.row_stack((x,y)) =  [[ 0  1  2  3]

 [47 83 38 53]]

np.column_stack((x,y)) =  [[ 0 47]

 [ 1 83]

 [ 2 38]

 [ 3 53]]

y =  [[76 24 15]

 [49 23 26]

 [30 43 30]

 [26 58 92]]

np.column_stack((x,y)) =  [[ 0 76 24 15]

 [ 1 49 23 26]

 [ 2 30 43 30]

 [ 3 26 58 92]]
```

Phương thức tile()

Đôi khi, bạn muốn hoặc phải tạo một ma trận mới bằng cách lặp lại một ma trận hiện có nhiều lần để tạo một ma trận mới với một hình dạng khác hoặc thậm chí là kích thước. Ví dụ lập lại ma trận được bôi đỏ để tạo thành ma trận như mảng dưới đây. Theo hàng 4 lần, cột lập 5 lần.

1	2	1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4	3	4

1	2	1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4	3	4

Cú pháp tile(x,reps), trong đó x là mảng cần lập, và reps là một tuple chỉ rõ hàng và cột cần lập bao nhiêu lần. Ví dụ:

```
import numpy as np
x =np.array([[1,2],[3,4]])
print "x = ", x
np.tile(x,(4,5))
print "np.tile(x,(4,5)) = ", np.tile(x,(4,5))


Output:


x =  [[1 2]

 [3 4]]

np.tile(x,(4,5)) =  [[1 2 1 2 1 2 1 2 1 2]

 [3 4 3 4 3 4 3 4 3 4]

 [1 2 1 2 1 2 1 2 1 2]

 [3 4 3 4 3 4 3 4 3 4]

 [1 2 1 2 1 2 1 2 1 2]

 [3 4 3 4 3 4 3 4 3 4]

 [1 2 1 2 1 2 1 2 1 2]

 [3 4 3 4 3 4 3 4 3 4]]
```

Kết luận

Qua bài này chúng tôi đã giới thiệu với các bạn thêm nhiều thao tác mới trên mảng, mà qua đó ảnh hưởng trực tiếp đến chiều của mảng. Sử dụng các phương thức flatten() hoặc ravel() khi bạn muốn chuyển mảng nhiều chiều về mảng chỉ có một chiều (hay làm phẳng mảng). Chú ý là flatten() trả về một copy của mảng gốc trong khi ravel() là trả về một view của mảng gốc. Reshape() đem đến nhiều tùy chọn hơn cho phép bạn tạo ra mảng mới có shape mới mà không thay đổi dữ liệu của nó. Ghép các mảng với phương thức concatenate(). Tăng chiều cho mảng đang tồn tại với numpy.newaxis. Xếp chồng các mảng theo hàng với row_stack() và theo cột với column_stack() hay tạo một ma trận mới bằng cách lặp lại ma trận hiện có nhiều lần với tile().

Trong bài tiếp theo chúng tôi sẽ trình bày chi tiết hơn về đối tượng kiểu dữ liệu ‘dtype’ – đối tượng này đã được sử dụng nhiều trong những bài học vừa qua.