Copies and views

When operating on NumPy arrays, it is possible to access the internal data buffer directly using a view without copying data around. This ensures good performance but can also cause unwanted problems if the user is not aware of how this works. Hence, it is important to know the difference between these two terms and to know which operations return copies and which return views.

The NumPy array is a data structure consisting of two parts: the contiguous data buffer with the actual data elements and the metadata that contains information about the data buffer. The metadata includes data type, strides, and other important information that helps manipulate the `ndarray` easily. See the Internal organization of NumPy arrays section for a detailed look.

View

It is possible to access the array differently by just changing certain metadata like stride and dtype without changing the data buffer. This creates a new way of looking at the data and these new arrays are called views. The data buffer remains the same, so any changes made to a view reflects in the original copy. A view can be forced through the `ndarray.view` method.

Copy

When a new array is created by duplicating the data buffer as well as the metadata, it is called a copy. Changes made to the copy do not reflect on the original array. Making a copy is slower and memory-consuming but sometimes necessary. A copy can be forced by using `ndarray.copy`.

Indexing operations

See also

Indexing on ndarrays

Views are created when elements can be addressed with offsets and strides in the original array. Hence, basic indexing always creates views. For example:

```
>>> x = np.arange(10)
>>> x
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> y = x[1:3]  # creates a view
>>> y
array([1, 2])
>>> x[1:3] = [10, 11]
>>> x
array([ 0, 10, 11,  3,  4,  5,  6,  7,  8,  9])
>>> y
array([10, 11])
```

Here, `y` gets changed when `x` is changed because it is a view.

Advanced indexing, on the other hand, always creates copies. For example:

```
>>> x = np.arange(9).reshape(3, 3)
>>> x
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> y = x[[1, 2]]
>>> y
array([[3, 4, 5],
       [6, 7, 8]])
>>> y.base is None
True
```

Here, `y` is a copy, as signified by the **base** attribute. We can also confirm this by assigning new values to `x[[1, 2]]` which in turn will not affect `y` at all:

```
>>> x[[1, 2]] = [[10, 11, 12], [13, 14, 15]]
>>> x
array([[ 0,  1,  2],
       [10, 11, 12],
       [13, 14, 15]])
>>> y
array([[3, 4, 5],
       [6, 7, 8]])
```

It must be noted here that during the assignment of `x[[1, 2]]` no view or copy is created as the assignment happens in-place.

Other operations

The **`numpy.reshape`** function creates a view where possible or a copy otherwise. In most cases, the strides can be modified to reshape the array with a view. However, in some cases where the array becomes non-contiguous (perhaps after a **`ndarray.transpose`** operation), the reshaping cannot be done by modifying strides and requires a copy. In these cases, we can raise an error by assigning the new shape to the shape attribute of the array. For example:

```
>>> x = np.ones((2, 3))
>>> y = x.T  # makes the array non-contiguous
>>> y
array([[1., 1.],
       [1., 1.],
       [1., 1.]])
>>> z = y.view()
>>> z.shape = 6
Traceback (most recent call last):
   ...
AttributeError: Incompatible shape for in-place modification. Use
`.reshape()` to make a copy with the desired shape.
```

Taking the example of another operation, **`ravel`** returns a contiguous flattened view of the array wherever possible. On the other hand, **`ndarray.flatten`** always returns a flattened copy of the array. However, to guarantee a view in most cases, `x.reshape(-1)` may be preferable.

How to tell if the array is a view or a copy

The **base** attribute of the ndarray makes it easy to tell if an array is a view or a copy. The base attribute of a view returns the original array while it returns `None` for a copy.

```
>>> x = np.arange(9)
>>> x
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>> y = x.reshape(3, 3)
>>> y
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> y.base  # .reshape() creates a view
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>> z = y[[2, 1]]
>>> z
array([[6, 7, 8],
       [3, 4, 5]])
>>> z.base is None  # advanced indexing creates a copy
True
```

Note that the `base` attribute should not be used to determine if an ndarray object is *new*; only if it is a view or a copy of another ndarray.

Previous
Universal functions (`ufunc`) basics

Next
Interoperability with NumPy