NumPy: Meaning of the axis parameter (0, 1, -1)

Posted: 2024-01-18 | Tags: Python, NumPy

In NumPy, functions like `np.sum()`, `np.mean()`, and `np.max()` have the `axis` parameter, which allows specifying the operation's target: the entire array, column-wise, row-wise, or other dimensions.

The meaning of the term "axis" in NumPy is explained in the official documentation's glossary as follows:

> **axis**
> Another term for an array dimension. Axes are numbered left to right; axis 0 is the first element in the shape tuple.
>
> *Glossary - axis — NumPy v1.26 Manual*

This article explains the meaning and usage of the `axis` parameter in NumPy.

Contents

The NumPy version used in this article is as follows. Note that functionality may vary between versions.

```
import numpy as np

print(np.__version__)
# 1.26.1
```

source: numpy_axis.py

In 2D array, `axis=0` operates column-wise, `axis=1` operates row-wise

In a two-dimensional array, `axis=0` operates column-wise, and `axis=1` operates row-wise.

For example, use `np.sum()` to calculate the sum.

```
a = np.arange(12).reshape(3, 4)
print(a)
# [[ 0  1  2  3]
#  [ 4  5  6  7]
#  [ 8  9 10 11]]

print(np.sum(a, axis=0))
# [12 15 18 21]

print(np.sum(a, axis=1))
# [ 6 22 38]
```

source: numpy_axis.py

The default is `axis=None`, which operates on the entire array.

```
print(np.sum(a))
# 66

print(np.sum(a, axis=None))
# 66
```

source: numpy_axis.py

An error is raised if an axis outside the array's dimensions is specified.

```
# print(np.sum(a, axis=2))
```

```
# AxisError: axis 2 is out of bounds for array of dimension 2
```

source: numpy_axis.py

The above example uses `np.sum()`, but the same applies to `np.mean()`, `np.max()`, `np.min()`, and so on.

- NumPy: Sum, mean, max, min for entire array, column/row-wise

`axis=-1` represents the last axis

You can use negative values for the `axis` parameter, which allows specifying an axis in reverse order from the last one. `-1` represents the last axis.

In a two-dimensional array, `axis=-1` is equivalent to `axis=1`, and `axis=-2` is equivalent to `axis=0`.

```
a = np.arange(12).reshape(3, 4)
print(a)
# [[ 0  1  2  3]
#  [ 4  5  6  7]
#  [ 8  9 10 11]]

print(np.sum(a, axis=-1))
# [ 6 22 38]

print(np.sum(a, axis=-2))
# [12 15 18 21]
```

source: numpy_axis.py

An error is raised if an axis outside the array's dimensions is specified.

```
# print(np.sum(a, axis=-3))
# AxisError: axis -3 is out of bounds for array of dimension 2
```

source: numpy_axis.py

Meaning of the `axis` parameter

For two-dimensional arrays, it is enough to remember that `axis=0` operates column-wise and `axis=1` operates row-wise, but let's consider the meaning of the `axis` parameter more generally.

Here, `np.sum()` is used as an example, but the same concept applies to other functions and methods like `np.mean()`.

In the case of 2D array

Consider a two-dimensional array with the shape `(2, 3)`.

```
a = np.arange(6).reshape(2, 3)
print(a)
# [[0 1 2]
#  [3 4 5]]
```

source: numpy_axis.py

`np.sum()` returns a one-dimensional array with the shape `(3,)` for `axis=0`, and `(2,)` for `axis=1`.

```
print(np.sum(a, axis=0))
# [3 5 7]

print(np.sum(a, axis=1))
# [ 3 12]
```

source: numpy_axis.py

`axis=0` aggregates along the first axis, and `axis=1` aggregates along the next axis, reducing the specified axis while preserving the other axis.

```
input  : (2, 3)
axis=0 : (_, 3) -> (3,)
axis=1 : (2, _) -> (2,)
```

`np.sum(axis=0)` and `np.sum(axis=1)` are equivalent to the following operations.

```
print(a[0, :] + a[1, :])
# [3 5 7]

print(a[:, 0] + a[:, 1] + a[:, 2])
# [ 3 12]
```

source: numpy_axis.py

`:` is used to select all elements along a given axis. The trailing `:` can be omitted, but is included here for clarity in the above example.

- NumPy: Slicing ndarray

In the case of 3D array

The same applies to three-dimensional and higher multi-dimensional arrays. Consider a three-dimensional array with the shape `(2, 3, 4)`.

- NumPy: Join arrays with np.concatenate, block, vstack, hstack, etc.

```
a = np.stack([np.ones((3, 4), int), np.full((3, 4), 10)])
print(a)
# [[[ 1  1  1  1]
#   [ 1  1  1  1]
#   [ 1  1  1  1]]
#
#  [[10 10 10 10]
#   [10 10 10 10]
#   [10 10 10 10]]]

print(a.shape)
# (2, 3, 4)
```

source: numpy_axis.py

`np.sum()` returns a two-dimensional array with the shape `(3, 4)` for `axis=0`, `(2, 4)` for `axis=1`, and `(2, 3)` for `axis=2`.

```
print(np.sum(a, axis=0))
# [[11 11 11 11]
#  [11 11 11 11]
```

```
#  [11 11 11 11]]

print(np.sum(a, axis=1))
# [[ 3  3  3  3]
#  [30 30 30 30]]

print(np.sum(a, axis=2))
# [[ 4  4  4]
#  [40 40 40]]
```

source: numpy_axis.py

This operation aggregates the input array along the axis specified by `axis`, reducing the specified axis while preserving the other axes.

```
input  : (2, 3, 4)
axis=0 : (_, 3, 4) -> (3, 4)
axis=1 : (2, _, 4) -> (2, 4)
axis=2 : (2, 3, _) -> (2, 3)
```

`np.sum(axis=0)`, `np.sum(axis=1)`, and `np.sum(axis=2)` are equivalent to the following operations.

```
print(a[0, :, :] + a[1, :, :])
# [[11 11 11 11]
#  [11 11 11 11]
#  [11 11 11 11]]

print(a[:, 0, :] + a[:, 1, :] + a[:, 2, :])
# [[ 3  3  3  3]
#  [30 30 30 30]]

print(a[:, :, 0] + a[:, :, 1] + a[:, :, 2] + a[:, :, 3])
# [[ 4  4  4]
#  [40 40 40]]
```

source: numpy_axis.py

When dealing with three-dimensional or higher multi-dimensional arrays, it may be more helpful to consider which axes to keep in the shape, rather than thinking in terms of rows, columns, depth, etc.

When specifying multiple values for `axis`

You can specify multiple values for the `axis` parameter with a tuple. The same concept applies here as well.

```
a = np.stack([np.ones((3, 4), int), np.full((3, 4), 10)])
print(a)
# [[[ 1  1  1  1]
#   [ 1  1  1  1]
#   [ 1  1  1  1]]
#
#  [[10 10 10 10]
#   [10 10 10 10]
#   [10 10 10 10]]]

print(a.shape)
# (2, 3, 4)

print(np.sum(a, axis=(0, 1)))
# [33 33 33 33]

print(np.sum(a, axis=(0, 2)))
# [44 44 44]

print(np.sum(a, axis=(1, 2)))
# [ 12 120]
```

source: numpy_axis.py

The relationship between the shapes of the input and output arrays can be considered as follows:

```
input       : (2, 3, 4)
axis=(0, 1) : (_, _, 4) -> (4,)
axis=(0, 2) : (_, 3, _) -> (3,)
axis=(1, 2) : (2, _, _) -> (2,)
```

Each is equivalent to the following operations.

```
print(
    a[0, 0, :] + a[0, 1, :] + a[0, 2, :] +
    a[1, 0, :] + a[1, 1, :] + a[1, 2, :]
)
# [33 33 33 33]

print(
    a[0, :, 0] + a[0, :, 1] + a[0, :, 2] + a[0, :, 3] +
```

```
    a[1, :, 0] + a[1, :, 1] + a[1, :, 2] + a[1, :, 3]
)
# [44 44 44]

print(
    a[:, 0, 0] + a[:, 0, 1] + a[:, 0, 2] + a[:, 0, 3] +
    a[:, 1, 0] + a[:, 1, 1] + a[:, 1, 2] + a[:, 1, 3] +
    a[:, 2, 0] + a[:, 2, 1] + a[:, 2, 2] + a[:, 2, 3]
)
# [ 12 120]
```

source: numpy_axis.py

`keepdims` maintains the dimensions of the output array

Functions and methods with the `axis` parameter also support the `keepdims` parameter. `keepdims=True` maintains the same number of dimensions in the output array as in the input array.

`np.sum()` is used as an example, but the same applies to other functions and methods like `np.mean()`.

The default output for a two-dimensional input array is one-dimensional, but `keepdims=True` returns a two-dimensional array.

```
a = np.ones((3, 4), int)
print(a)
# [[1 1 1 1]
#  [1 1 1 1]
#  [1 1 1 1]]

print(a.shape)
# (3, 4)

print(np.sum(a, axis=1))
# [4 4 4]

print(np.sum(a, axis=1).shape)
# (3,)

print(np.sum(a, axis=1, keepdims=True))
# [[4]
#  [4]
#  [4]]
```

```
print(np.sum(a, axis=1, keepdims=True).shape)
# (3, 1)
```

source: numpy_keepdims.py

With `keepdims=True`, the output array is correctly broadcast with the input array.

In NumPy, during binary operations, array shapes are automatically aligned through broadcasting where possible.

- NumPy: Broadcasting rules and examples

In operations involving the output array and the input array (or any array with the same shape as the input), using `axis=1` with default setting may result in an error. However, `keepdims=True` ensures proper broadcasting.

```
# print(a + np.sum(a, axis=1))
# ValueError: operands could not be broadcast together with shapes (3,4) (3,)

print(a + np.sum(a, axis=1, keepdims=True))
# [[5 5 5 5]
#  [5 5 5 5]
#  [5 5 5 5]]
```

source: numpy_keepdims.py

For `axis=0`, the default setting correctly broadcasts as well, but `keepdims=True` is also acceptable.

```
print(a + np.sum(a, axis=0))
# [[4 4 4 4]
#  [4 4 4 4]
#  [4 4 4 4]]

print(a + np.sum(a, axis=0, keepdims=True))
# [[4 4 4 4]
#  [4 4 4 4]
#  [4 4 4 4]]
```

source: numpy_keepdims.py

For operations involving broadcasting to the input array, it may be safer to set `keepdims=True` to avoid mistakes.

The same applies to three-dimensional and higher multi-dimensional arrays. Regardless of the value specified for `axis`, `keepdims=True` ensures that the output array is correctly broadcast with the input array.

```
a = np.ones((2, 3, 4), int)
print(a)
# [[[1 1 1 1]
#   [1 1 1 1]
#   [1 1 1 1]]
#
#  [[1 1 1 1]
#   [1 1 1 1]
#   [1 1 1 1]]]

print(np.sum(a, axis=(0, 2)))
# [8 8 8]

print(np.sum(a, axis=(0, 2), keepdims=True))
# [[[8]
#   [8]
#   [8]]]

# print(a + np.sum(a, axis=(0, 2)))
# ValueError: operands could not be broadcast together with shapes (2,3,4) (3,)

print(a + np.sum(a, axis=(0, 2), keepdims=True))
# [[[9 9 9 9]
#   [9 9 9 9]
#   [9 9 9 9]]
#
#  [[9 9 9 9]
#   [9 9 9 9]
#   [9 9 9 9]]]
```

source: numpy_keepdims.py

Related Categories

- Python
- NumPy

Related Articles

- NumPy: Create an empty array (np.empty, np.empty_like)
- Matrix operations with NumPy in Python
- NumPy: Insert elements, rows, and columns into an array with np.insert()
- Convert between NumPy array and Python list
- NumPy: Save and load arrays in npy and npz files
- NumPy: Delete rows/columns from an array with np.delete()
- NumPy: append() to add values to an array
- NumPy: Calculate cumulative sum and product (np.cumsum, np.cumprod)
- NumPy: Calculate the absolute value element-wise (np.abs, np.fabs)
- NumPy: Round array elements (np.round, np.around, np.rint)
- NumPy: arange() and linspace() to generate evenly spaced values
- Check NumPy version: np.version
- Binarize image with Python, NumPy, OpenCV
- NumPy: Rotate array (np.rot90)
- List of NumPy articles