

行前準備：

① NumPy 陣列快速上手

- 0-0 陣列的 shape 與軸、階、維度
- 0-1 NumPy 陣列的屬性與型別轉換
- 0-2 建立 NumPy 陣列 (array)
- 0-3 陣列切片 (slicing)：截取陣列片段
- 0-4 陣列的重塑與轉置
- 0-5 陣列間對應元素 (element-wise) 的運算
- 0-6 陣列擴張 (broadcasting)
- 0-7 組合多個索引來批次設值或取值
- 0-8 沿著陣列的某一軸做運算

前言

歡迎來到 `tf.keras` 的深度學習世界，
由於 `tf.keras` 是使用張量與 `NumPy` 陣列來儲存陣列資料，
而張量其實就是 `NumPy` 陣列的擴充，
因此它們在資料的操作及運算上都是相通的。

0-0

陣列的 shape 與軸、階、維度

在 Python 中可以用 list 或 tuple 來儲存陣列資料,而陣列資料可以是單層的,也可以是多層的。

NumPy 為了提高陣列的運算效率, 要求每個 NumPy 陣列都必須有固定的結構,並用 shape (形狀) 來描述其結構：

0-0

陣列的 shape 與軸、階、維度

- 1 層且內含 3 個元素的陣列, 其 shape 為 (3,), 如 [1, 2, 3]。
- 2 層的 2×3 陣列, 其 shape 為 (2, 3), 如 [[1,2,3], [4,5,6]]。

0-0

陣列的 shape 與軸、階、維度

- 3 層的 $2 \times 3 \times 2$ 陣列, 其 shape 為 $(2, 3, 2)$,
如 $[[[1,2],[3,4],[5,6]], [[2,3],[4,5],[6,7]]]$ 。
- 更多層的陣列, 則 shape 的 tuple 中會有更多的元素,
如 5 層的 $2 \times 3 \times 2 \times 4 \times 3$ 陣列, shape 為 $(2, 3, 2, 4, 3)$ 。

0-0

陣列的 shape 與軸、階、維度

通常 shape 有幾個元素, 即稱為幾軸 (axis) 或幾階 (rank), 而每一軸的元素個數則稱為維度 (dimention)。

一般會將單軸的陣列稱為向量, 而 2 軸的陣列則稱為矩陣。
因此 shape (3,) 的陣列就稱為「3 維向量」,
shape (2, 3) 的陣列則可稱為「2×3 矩陣」。

0-0

陣列的 shape 與軸、階、維度

3D 向量和 3D 陣列這兩者是有所不同的。

如果用在向量, D 是代表向量的維度 (Dimension),
因此 shape 為 (3,) 的向量就是一個 3D 向量, 有 3 個元素。
如果用在陣列, 則 D 是代表陣列的軸數 (或階數)。

0-0

陣列的 shape 與軸、階、維度

所以 D 的意思要看是用在向量還是陣列而有所不同。

為避免混淆, 本書都以 D 來代表軸,

1D 就是 1 軸陣列 (也就是向量),

2D 就是 2 軸陣列 (就是矩陣), 依此類推。

0-0

陣列的 shape 與軸、階、維度

● NumPy 陣列的優點及特色

NumPy 陣列和 Python 的 list 主要有以下 3 點差異：

1. NumPy 陣列佔用空間較小, 且存取速度較快。
2. NumPy 陣列中所有元素的型別都必須相同, 而 list 或 tuple 無此限制。
3. NumPy 陣列中每軸的元素數目必須和 shape 的維度完全一致。list 或 tuple 則無此限制。

0-1

NumPy 陣列的 屬性與型別轉換

```
>>> import numpy as np ← 使用前要先匯入 NumPy 套件
>>> x = np.zeros((3, 5, 2)) ← 建立一個形狀為 (3,5,2) 且元素均為 0 的陣列
>>> x.ndim ← 顯示軸數屬性
3 ← 有 3 個軸 (為 3D 陣列)
>>> x.shape ← 顯示形狀屬性
(3, 5, 2) ← 為 3×5×2 的陣列
>>> x.dtype ← 顯示型別屬性
dtype('float64') ← 型別為 float64 (64 位元的浮點數)
```

0-1

NumPy 陣列的 屬性與型別轉換

由形狀也可看出軸數，因為 `len(x.shape)` 的結果也是 3。

幾軸通常就稱為幾 D，而 2D 陣列就是矩陣；

1D 陣列則為向量；而 0D 陣列就是純量的意思。

`shape (3,)` 和 `(3,1)` 是不同的，後者為 2D 矩陣。

0-1

NumPy 陣列的 屬性與型別轉換

常見的陣列元素型別有 `int8`、`uint8`、`uint64`、`float32`、`float64`、`bool` (布林值) 等。使用 `astype()` 可以將陣列內容轉出為指定的型別：

```
>>> x = np.zeros(5)
>>> y = x.astype('uint8')  ← 轉出為 uint8 並指定給 y 變數 (不會
>>> y.dtype                改變原資料, 所以我們稱為 "轉出")
dtype('uint8')             ← 已轉出為 uint8 型別
>>> x.dtype
dtype('float64')           ← 原資料不受影響
```

0-2 建立 NumPy 陣列 (array)

建立 NumPy 陣列的幾種較常用方法：

```
>>> import numpy as np
>>> x = np.array([[1,2], [3,4]])  ← 用實際的資料 (例如串列
>>> x                                或 tuple) 建立陣列
array([[1, 2], [3, 4]])
    ↑
    | 前面有這個表示其為 NumPy 陣列 (ndarray) 物件
```

```
>>> x = np.zeros((2, 3, 2))  ← 依指定的 shape 建立全部為 0 的陣列
>>> x
array([[ [0., 0.],
         [0., 0.],
         [0., 0.]],
       [ [0., 0.],
         [0., 0.],
         [0., 0.] ]])
```

← shape 由左而右, 就表示陣列由外而內：
最外層有 2 個大元素, 每個大元素內有 3 個中元素,
每個中元素內又有 2 個小元素。

0-2 建立 NumPy 陣列 (array)

觀看陣列資料的小技巧

最內層的元素
前面只有 1 個 '['


看這裡可知是倒
數第幾層的開始

看這裡就知道
全部有幾層

最外層

最內層

```
array([[[[0., 0.],  
        [0., 0.],  
        [0., 0.]],  
       [[0., 0.],  
        [0., 0.],  
        [0., 0.]]])
```



The diagram illustrates the structure of a 3D NumPy array. The array is represented as `array([[[[0., 0.], [0., 0.], [0., 0.]], [[0., 0.], [0., 0.], [0., 0.]]])`. Annotations include: '最內層的元素 前面只有 1 个 '['' pointing to the first '[' of the innermost list; '看這裡可知是倒數第幾層的開始' pointing to the '[' of the second-to-last list; '看這裡就知道全部有幾層' pointing to the '[' of the outermost list; '最外層' pointing to the outermost '['; and '最內層' pointing to the innermost '['. A cartoon character is shown pointing at the array on the right side of the whiteboard.

```
>>> x = np.zeros(2, dtype='uint8')  
>>> x  
array([0, 0], dtype=uint8)
```

← 建立形狀為 (2,) 的 uint8 全 0 陣列,
第 1 個參數為 shape, 若為 1D 可只傳
入維度, 例如傳入 2 和 (2,) 是一樣的

```
>>> x = np.ones((2, 3))
```

← 用 ones() 依 shape 建立全為 1 的陣列,
參數用法同 zeros()

```
>>> x = np.full((2, 3), 7)
```

← 用 full() 依 shape 建立全為 7 的陣列,
第 1 個參數用法同 zeros(), 第 2 個參
數是指定元素的值

```
>>> x = np.arange(5)  
>>> x  
array([0, 1, 2, 3, 4])
```

← 用 arange() 建立陣列,
參數用法同 Python 的 range()

```
>>> y = np.copy(x)
```

← 用 copy() 將 x 複製一份

```
>>> y = np.zeros_like(x)
```

← 用 zeros_like() 依 x 的形狀建立全 0 陣列,
相當於 np.zeros(x.shape)

```
>>> y = np.ones_like(x)
```

← 用 ones_like() 依 x 的形狀建立全 1 陣列,
相當於 np.ones(x.shape)

0-3

陣列切片 (slicing) : 截取陣列片段

NumPy 陣列的切片和 Python 的 list 切片類似,
 $m:n$ 是由 m 到 n 但不含 n 。

若 m 省略, 表示由最前面開始取,

若 n 省略, 表示取到 (包含) 最後一個,

如果都省略 (只寫冒號) 就表示要取全部。

若 m 或 n 為負數, 則表示倒數第幾個。

0-3

陣列切片 (slicing) : 截取陣列片段

```
>>> x = np.array([[1, 2, 3],  
                  [4, 5, 6],  
                  [7, 8, 9]])
```

← 先建一個陣列

```
>>> x[1, 2] ← 以逗號分隔索引 (讀取第 0 軸  
6           索引 1、第 1 軸索引 2 的元素)
```

```
>>> x[1][2] ← 也可以用 [] 分隔索引 (Python 的寫法)  
6
```

```
>>> x  
array([[1, 2, 3],  
       [4, 5, 6],  
       [7, 8, 9]])
```

```
>>> x[0:2, 1:3] ← 第 0 軸取 0~1 的範圍,  
array([[2, 3],  
       [5, 6]]) 第 1 軸取 1~2 的範圍
```

第 1 軸索引 2

1	2	3
4	5	6
7	8	9

第 0 軸
索引 1 →

第 1 軸取 1~2

第 0 軸
取 0~1

1	2	3
4	5	6
7	8	9

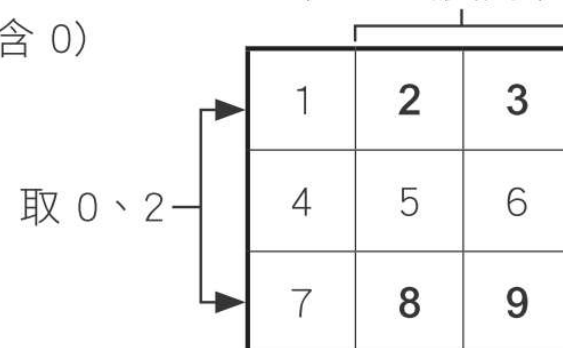
0-3

陣列切片 (slicing) : 截取陣列片段

```
>>> x[:,::2,  
      2:0:-1]  
array([[3, 2],  
       [9, 8]])
```

← 第 0 軸間隔 2 取樣：取出索引 0 和索引 2
← 第 1 軸反向取樣：由 2 取到 1 (不含 0)

取 2~1 (反向取)



0-3

陣列切片 (slicing) : 截取陣列片段

如果省略最後面的幾軸不寫, 則表示那些軸全部都要,
相當於在該軸使用「:」來表示:

```
>>> x[:2]  ← 第 0 軸寫 :2 表示取 0~1, 第 1 軸沒寫, 表示取全部  
array([[1, 2, 3],  
       [4, 5, 6]])
```

0-4 陣列的重塑與轉置

- 陣列重塑 (reshape) : 改變陣列形狀

陣列重塑 (reshape) 就是改變陣列的形狀,
但元素的總數不會改變 :

```
>>> x = np.array([[[1,2], [3,4]],  
                  [[5,6], [7,8]]]) ]—— 建立 shape (2,2,2) 的陣列  
>>> y = np.reshape(x, (2, 4)) ←—— 轉出為 shape (2,4) 的陣列  
>>> y  
array([[1, 2, 3, 4],  
       [5, 6, 7, 8]]) ←—— 已將元素依序複製到新的 shape 中
```

0-4 陣列的重塑與轉置

- 矩陣轉置 (transpose) :
將矩陣的直行與橫列交換

矩陣轉置就將矩陣的直行與橫列交換：

```
>>> x = np.array([[1,2,3], [4,5,6]])
>>> x
array([[1, 2, 3],
       [4, 5, 6]]) ]— shape(2, 3)
>>> np.transpose(x) ← 轉置的結果會儲存到新的陣列中傳回 (原陣列 x 的內容不變)
array([[1, 4],
       [2, 5],
       [3, 6]]) ]— shape(3, 2)
```

2 個 **shape** 相同的陣列, 可直接進行加減乘除或比較等運算, 此時會將同位置的元素進行運算, 並將運算結果存放到新的陣列中, 其 **shape** 會和原來的陣列相同 :

```
>>> np.array([[1,2],[3,4]]) + np.array([[1,2],[3,4]])  ┌─ 矩陣逐元素相加
array([[2, 4], [6, 8]])

>>> np.array([1,2]) * np.array([3,4])  ┌─ 向量逐元素相乘
array([3, 8])

>>> np.array([1,3]) > np.array([2,2])  ── 向量逐元素比較大小
array([False,  True])
      ↑       ↑
      1>2 為 False  3>2 為 True
```

0-6 陣列擴張 (broadcasting)

如果要運算的 2 個陣列 `shape` 不相同,
那 NumPy 會試著以「複製現有資料」的方式,
擴張任一邊或二邊陣列的 `shape` 來使其形狀相同,
此自動擴張 `shape` 的方式即稱為陣列擴張。

```
>>> np.array([[1,2],[3,4]]) + np.array([1,2]) ← shape (2,2) + (2,)
array([[2, 4],
       [4, 6]])
```

1	2
3	4

 +

1	2
1	2

 =

2	4
4	6

第 2 個陣列會先由 shape (2,) 擴張成 (2, 2) 然後再相加

```
>>> np.array([[1],[2]]) + np.array([3,4]) ← shape (2,1) + (2,)
array([[4, 5],
       [5, 6]])
```

1	1
2	2

 +

3	4
3	4

 =

4	5
5	6

第 1 個陣列會由 shape (2,1) 擴張成 (2, 2)
第 2 個陣列會由 shape (2,) 擴張成 (2,2)

0-6 陣列擴張 (broadcasting)

- 陣列擴張的詳細規則

陣列擴張會依照底下 2 個步驟進行自動擴張：

```
a = [[[1], [2]]] # shape: (1, 2, 1)
b = [ 3, 4]      # shape: (2,)
```

1.

```
a = [[[1], [2]]] # shape: (1, 2, 1)
b = [[[3, 4]]]   # shape: (2,) ➡ (1, 1, 2) ← 由 1 軸擴張成 3 軸
```

0-6 陣列擴張 (broadcasting)

2.

```
a = [[[1, 1], [2, 2]]] # shape: (1, 2, 1) ➡ (1, 2, 2)
b = [[[3, 4], [3, 4]]] # shape: (1, 1, 2) ➡ (1, 2, 2)
```

因此 $a + b$ 的運算結果為 $[[[4, 5], [5, 6]]]$ ，
其 shape 為 $(1, 2, 2)$ 。

0-7

組合多個索引來 批次設值或取值

可將多個索引值組合為 list、tuple、或 NumPy 陣列等，
然後用它來做為其他陣列的索引，以進行批次設值或取值：

```
>>> x = np.array([0, 1, 2, 3, 4])
>>> idx = [1, 3]

>>> x[idx] = 7  ← 將第 1、3 個元素設為 7
>>> x
array([0, 7, 2, 7, 4])
```

0-7

組合多個索引來 批次設值或取值

```
>>> x[idx] = (8, 9) ← 將第 1、3 個元素設為 8、9 (此時二邊的元素數量要相等才行)
>>> x
array([0, 8, 2, 9, 4])

>>> y = x[idx] ← 取出第 1、3 個元素值給 y
>>> y
array([8, 9])

>>> y[[1, 0, 1]] ← 元素也可以重複取 (第 1 個元素取了 2 次)
array([9, 8, 9])
```

0-7

組合多個索引來 批次設值或取值

此外還可用 True 及 False 來一一指定哪些元素要被存取：

```
>>> x = np.array([0, 1, 2])
>>> idx = np.array([True, False, True])
>>> x[idx]  ← 只取第 0、2 個元素 (索引位置為 True 的元素)
array([0, 2])

>>> x[idx] = 7  ← 將第 0、2 個元素設為 7
>>> x
array([7, 1, 7])
```

0-8 沿著陣列的某一軸做運算

有時候會希望能沿著陣列的某一軸做運算：

	國文	英文	數學
小明	85	90	77
小美	88	82	93

沿著第 1 軸 (橫列) 可計算每個學生的最高成績：[90, 93]

沿著第 0 軸 (直行) 可計算每個學科的最高成績：[88, 90, 93]

0-8 沿著陣列的某一軸做運算

再以實際的程式來看，當用 `np.max()` 取陣列的最大值時，若未指定軸：

```
>>> a = np.array([[1, 2],
                  [3, 4]])
>>> np.max(a) ← 對所有元素取最大值
4
```

但也可以加上 `axis=n` 參數來沿著第 `n` 軸取最大值：

```
>>> np.max(a, axis=0) ← 沿著第 0 軸取最大值
array([3, 4])      # = [max(1,3), max(2,4)]

>>> np.max(a, axis=1) ← 沿著第 1 軸取最大值
array([2, 4])      # = [max(1,2), max(3,4)]
```

0-8 沿著陣列的某一軸做運算

「沿著第 n 軸做運算」意思就是沿著第 n 軸取出每個向量來做運算。

實際取出的第一個向量, 就是由其他軸索引均為 0、第 n 軸索引為 0、1、2 的元素所組成的向量;

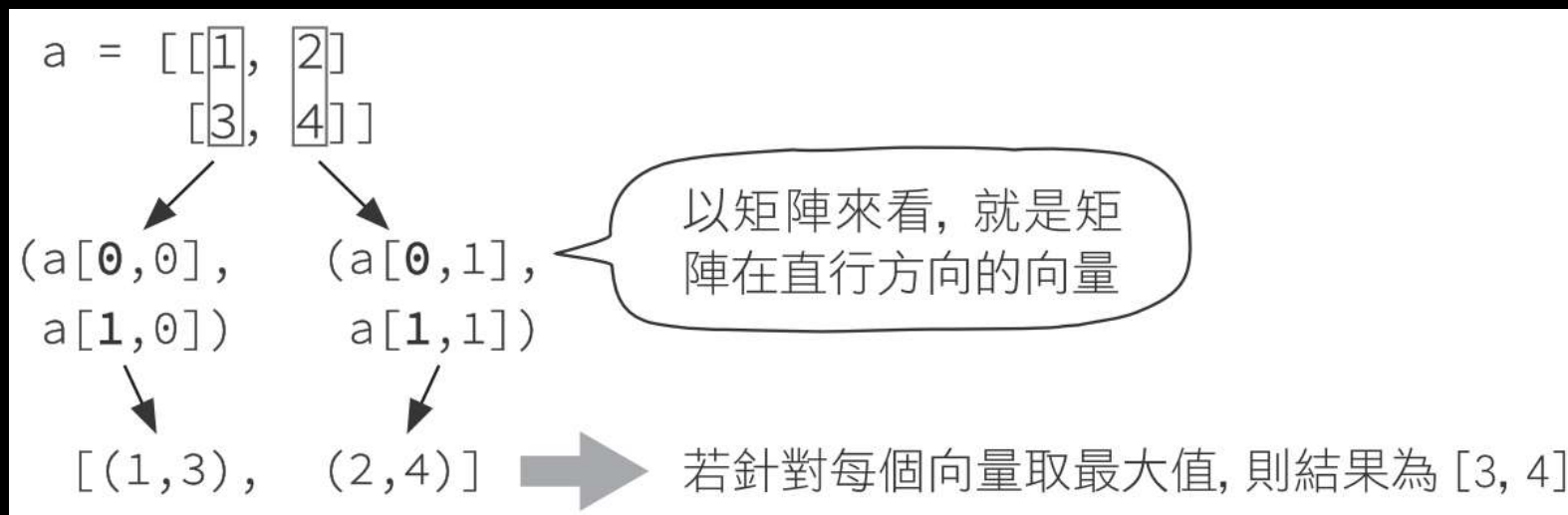
所取出的第二個向量, 則是將最後一個非 n 軸的索引加 1, 然後用同樣的方式取出的向量, 以此類推...

0-8 沿著陣列的某一軸做運算

當最後一個非 n 軸的索引加滿時則歸 0,
並將其前一個非 n 軸的索引加 1, 然後繼續取向量,
直到所有的非 n 軸的索引均取過向量為止。

0-8 沿著陣列的某一軸做運算

- 沿著第 0 軸取出每一個向量，先取第 1 軸索引為 0、第 0 軸索引為 0、1 的元素來組成第一個向量，然後將第 1 軸索引加 1 再用同樣方法取出第二個向量：



0-8 沿著陣列的某一軸做運算

- 沿著第 1 軸取出每一個向量，則是先取第 0 軸索引為 0、第 1 軸索引為 0、1 的元素來組成第一個向量，然後將第 0 軸索引加 1 再用同樣方法取出第二個向量：

$a = \begin{bmatrix} [1, 2] \\ [3, 4] \end{bmatrix} \rightarrow (a[0,0], a[0,1]) \rightarrow [(1,2),$
 $\quad \quad \quad \rightarrow (a[1,0], a[1,1]) \rightarrow (3,4)]$

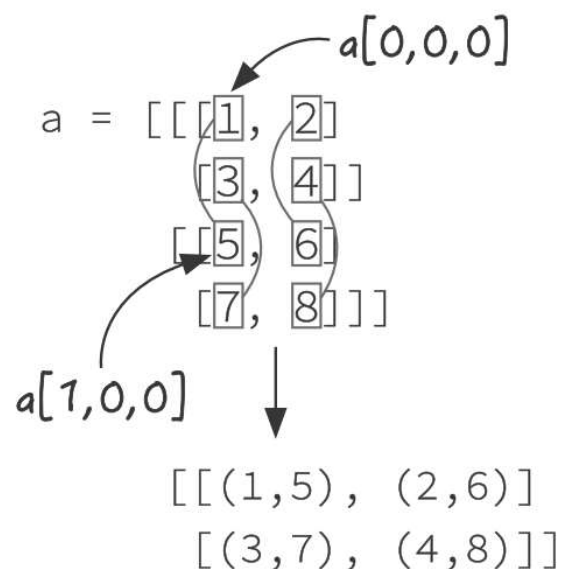
以矩陣來看，就是矩陣
在橫列方向的向量



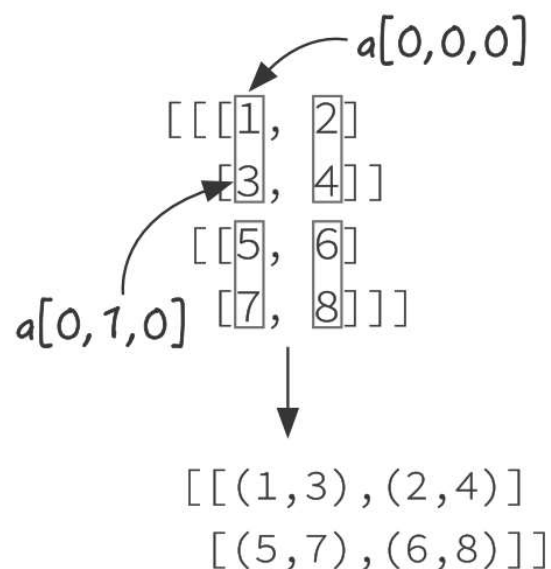
若針對每個向量取最大值，則結果為 [2, 4]

0-8 沿著陣列的某一軸做運算

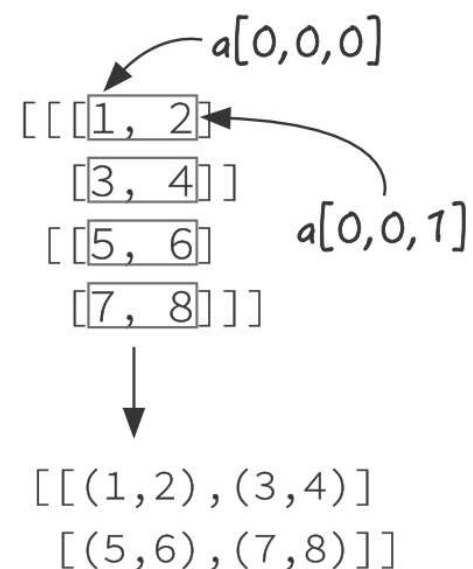
● 沿第 0 軸取向量：



● 沿第 1 軸取向量：



● 沿第 2 軸取向量：



0-8 沿著陣列的某一軸做運算

以上若執行 `np.max(a, axis=i)` 沿著第 i 軸對每個向量取最大值, 則 $i = 0$ 、 1 、 2 時的結果如下：

$i = 0$: $\begin{bmatrix} [5, 6], \\ [7, 8] \end{bmatrix}$

$i=1$: $\begin{bmatrix} [3, 4], \\ [7, 8] \end{bmatrix}$

$i=2$: $\begin{bmatrix} [2, 4], \\ [6, 8] \end{bmatrix}$

0-8 沿著陣列的某一軸做運算

當沿著某一軸的向量做運算時，由於向量運算的結果為純量，因此該軸在運算結果中會消失：

$a = \begin{bmatrix} [1, 2] \\ [3, 4] \\ [5, 6] \end{bmatrix}$ \rightarrow $\begin{bmatrix} \max(1, 2) \\ \max(3, 4) \\ \max(5, 6) \end{bmatrix}$ \rightarrow $\begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}$

0-8 沿著陣列的某一軸做運算

同理, `np.max(a, axis=0)` 的結果 `shape` 會變成 `(3, 2)`,
而 `np.max(a, axis=1)` 的結果 `shape` 會變成 `(1, 2)`。

未來如果看到類似 `np.max(a, axis=(0,2))` 的寫法,
則可把它看成是連續對多個軸取最大值。