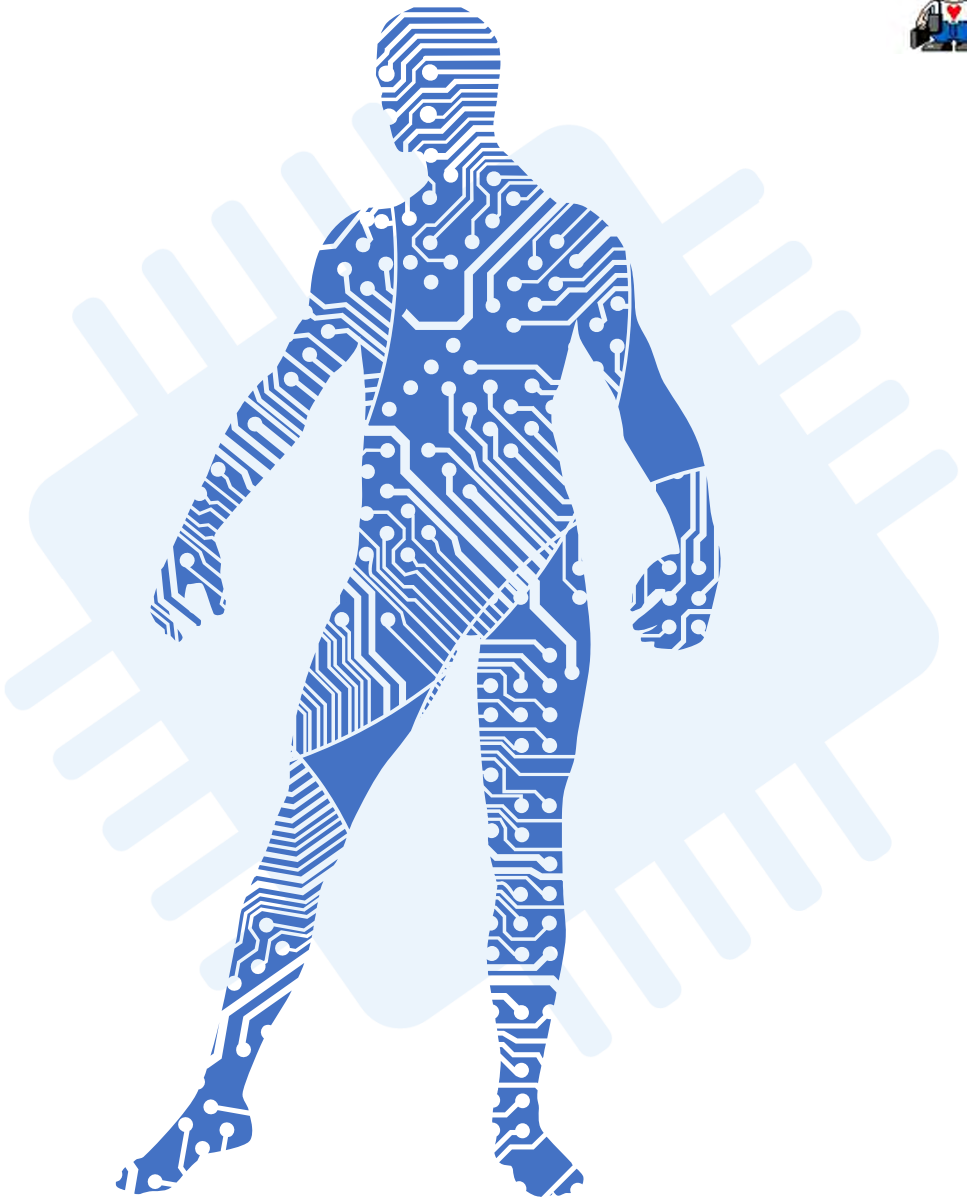
自然語言處理

第 4 章 PyTorch 張量操作

講師：紀俊男

本章大綱

- PyTorch 簡介
- 張量簡介
- 張量建立
- 張量本身資訊之取得
- 張量元素的讀寫
- 張量樣本點的產生
- 張量基本運算



PyTorch 簡介

範例完整原始碼：
https://lurl.cc/J9qSZM

PyTorch 特色

- **豐富的模型與優化器**
  - PyTorch 在自然語言領域中，提供比其它框架更豐富的**模型**與**優化器**。
- **自動微分（Autograd）**
  - 計算神經網路各個參數梯度時，會按照當時佈建的神經網路架構求導，而非將微分求導過程，**寫死**在模型內。
- **動態神經網路**
  - 有機會能在**執行時期**，變更神經網路的**層數**、**節點**、與**架構**。
- **資料交換**
  - 能與 Python, NumPy, Pandas 等函式庫的資料結構，輕鬆交換資料。

安装 PyTorch

- 安装指令
  - pip install torch

- 確認安装成功

```
1  import torch
2
3  print(torch.__version__)
```

PyTorch

隨堂練習

- 請新建一個 Colab 頁面，假設命名為 PyTorchDemo.ipynb。

- 請在該檔案內，撰寫下列程式碼，並執行看看：

```
1  !pip install torch
2
3  import torch
4  print(torch.__version__)
```

- 說說看，您目前的 PyTorch 版本是第幾版？

檢查是否有 GPU 可供使用

- 程式碼

控制將來使用哪種
「算力」的重要變數

```
1  if torch.cuda.is_available():
2      device = torch.device("cuda")
3      print("Using GPU")
4  else:
5      device = torch.device("cpu")
6      print("Using CPU")
```

隨堂練習

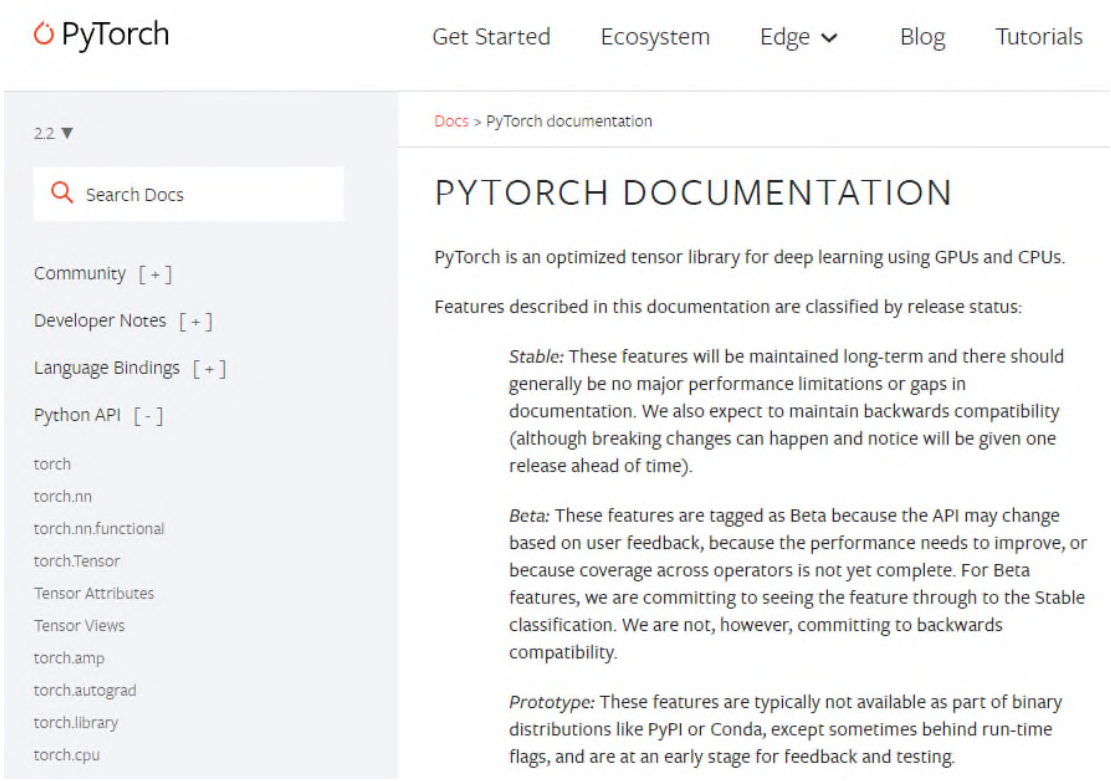
- 繼續使用前一個 Colab 頁面 PyTorchDemo.ipynb。
- 在該檔案內，撰寫下列程式碼：

```
if torch.cuda.is_available():
  device = torch.device("cuda")
  print("Using GPU")
else:
  device = torch.device("cpu")
  print("Using CPU")
```

- 將您的 Colab 頁面，分別調整成 CPU 與 GPU，，並把整個程式碼執行看看。
- 觀察調整成 CPU 與 GPU 時，畫面上顯示的文字有何不同？

取得 PyTorch 求助訊息

- PyTorch 最新版使用手冊：



隨堂練習

- 請點擊下列連結
  - https://pytorch.org/docs/stable/index.html

- 簡單瀏覽一下裡面的內容。

AI 張量簡介

何謂「張量（Tensors）」

- PyTorch 中，儲存**數據**的**資料結構**。

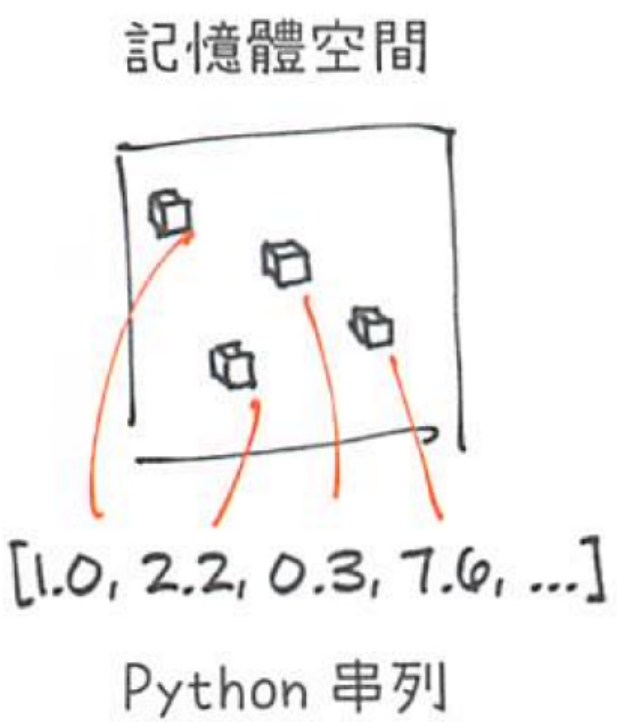
	A	B	C	D
1	國別	年齡	薪資	是否購買
2	1	44	72000	0
3	2	27	48000	1
4	3	30	54000	0
5	2	38	61000	0
6	3	40	63777.78	1
7	1	35	58000	1
8	2	38.78	52000	0
9	1	48	79000	1
10	3	50	83000	0
11	1	37	67000	1

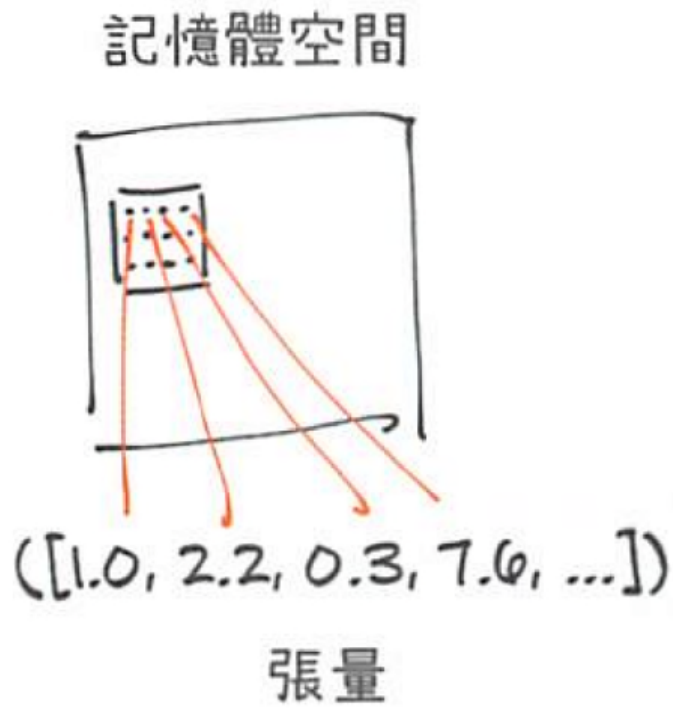
```
tensor([[1.0000e+00, 4.4000e+01, 7.2000e+04, 0.0000e+00],
        [2.0000e+00, 2.7000e+01, 4.8000e+04, 1.0000e+00],
        [3.0000e+00, 3.0000e+01, 5.4000e+04, 0.0000e+00],
        [2.0000e+00, 3.8000e+01, 6.1000e+04, 0.0000e+00],
        [3.0000e+00, 4.0000e+01, 6.3778e+04, 1.0000e+00],
        [1.0000e+00, 3.5000e+01, 5.8000e+03, 1.0000e+00],
        [1.0000e+00, 3.8780e+01, 5.2000e+04, 0.0000e+00],
        [1.0000e+00, 4.8000e+01, 7.9000e+04, 1.0000e+00],
        [3.0000e+00, 5.0000e+01, 8.3000e+04, 0.0000e+00],
        [1.0000e+00, 3.7000e+01, 6.7000e+04, 1.0000e+00]])
```

預設資料型態：32 位元寬度的浮點數

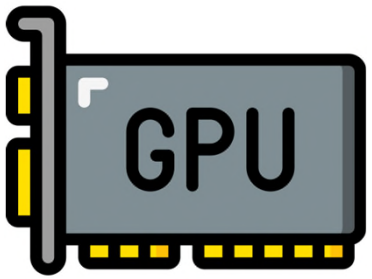
與 Python List 的不同點

- 張量所霸佔的記憶體，預設是「連續」的

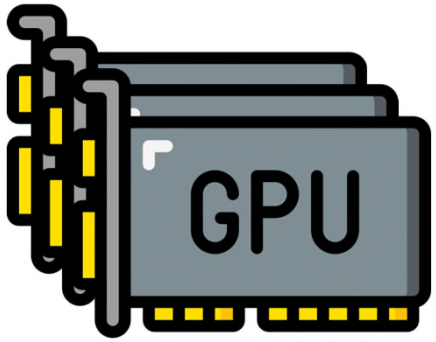




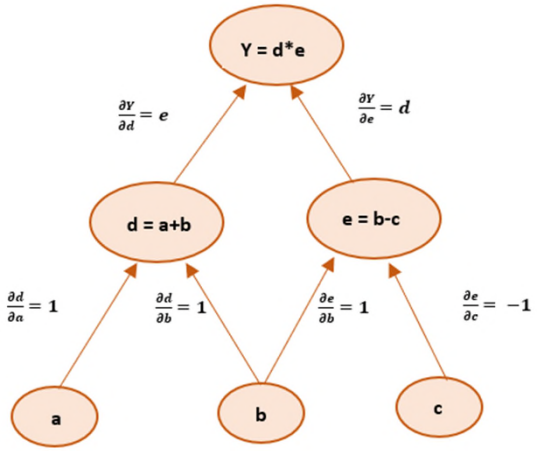
比 NumPy NDArray 優秀的地方



能夠搬運到 GPU 運算



能夠在多個設備上
平行運算



能以「運算圖」追蹤
計算過程，並加以優化

常見的張量資料型態名稱（dtype）

	整數（Integers）	浮點數（Floats）	布林值（Boolean）
dtype Name (for CPU/GPU)	torch.uint8 / torch.int8 (unsigned / signed 8 bits) torch.short / torch.int16 (signed 16 bits) torch.int / torch.int32 (signed 32 bits) torch.long / torch.int64 (signed 64 bits)	torch.half / torch.float16 torch.float / torch.float32 torch.double / torch.float64	torch.bool
CPU Tensors	torch.ByteTensor / torch.CharTensor torch.ShortTensor torch.IntTensor torch.LongTensor	torch.HalfTensor (16 bits) torch.FloatTensor (32 bits) torch.DoubleTensor (64 bits)	torch.BoolTensor
GPU Tensors	torch.cuda.ByteTensor / torch.cuda.CharTensor torch.cuda.ShortTensor torch.cuda.IntTensor torch.cuda.LongTensor	torch.cuda.HalfTensor (16 bits) torch.cuda.FloatTensor (32 bits) torch.cuda.DoubleTensor (64 bits)	torch.cuda.BoolTensor

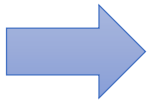
張量建立

建立一般張量

- torch.Tensor()：強制建立 32 位元浮點數張量

- 一維浮點數張量

```
1  import tensor
2
3  tensor_1D = torch.Tensor([1, 2, 3])
4  print("Tensor 1D:", tensor_1D)
5  print(tensor_1D.type())
```

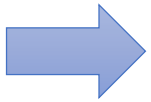


[1., 2., 3.]
torch.FloatTensor

32 bits CPU 浮點數

- 二維浮點數張量

```
1  import tensor
2
3  tensor_2D = torch.Tensor([[1, 2, 3], [4, 5, 6]])
4  print("Tensor 2D:", tensor_2D)
5  print(tensor_2D.type())
```



[[1., 2., 3.]
 [4., 5., 6.]]
torch.FloatTensor

隨堂練習：以 torch.Tensor() 建立張量

- 請用下列程式碼，試著建立一維＆二維張量，並將它印出：

```
1  import tensor
2
3  tensor_1D = torch.Tensor([1, 2, 3])
4  print("Tensor 1D:", tensor_1D)
5  print(tensor_1D.type())
6
7  tensor_2D = torch.Tensor([[1, 2, 3], [4, 5, 6]])
8  print("Tensor 2D:", tensor_2D)
9  print(tensor_2D.type())
```

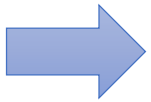
```
Tensor 1D: tensor([1., 2., 3.])
torch.FloatTensor
Tensor 2D: tensor([[1., 2., 3.],
        [4., 5., 6.]])
torch.FloatTensor
```

建立一般張量

- torch.tensor()：依照傳入數據的資料型態，建立對應的張量

- 一維整數張量

```
1  import tensor
2
3  tensor_1D = torch.tensor([1, 2, 3])
4  print("Tensor 1D:", tensor_1D)
5  print(tensor_1D.type())
```

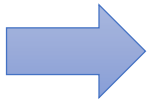


[1, 2, 3]
torch.LongTensor

64 bits CPU 長整數

- 二維整數張量

```
1  import tensor
2
3  tensor_2D = torch.tensor([[1, 2, 3], [4, 5, 6]])
4  print("Tensor 2D:", tensor_2D)
5  print(tensor_2D.type())
```



[[1, 2, 3]
 [4, 5, 6]]
torch. LongTensor

建立一般張量

- torch.tensor(...dtype=)：指定建立的張量資料型態

```
1  import tensor
2
3  tensor_1D = torch.tensor([1, 2, 3], dtype=torch.float)
4  print("Tensor 1D:", tensor_1D)
5  print(tensor_1D.type())
```

[1., 2., 3.]
torch.FloatTensor

隨堂練習：以 torch.tensor() 建立張量

- 請用下列程式碼，試著建立一維＆二維張量，並將它印出：

```python
import tensor

# 依照傳入數據的資料型態，建立對應的張量
tensor_1D = torch.tensor([1, 2, 3])
print("Tensor 1D:", tensor_1D)
print(tensor_1D.type())

tensor_2D = torch.tensor([[1, 2, 3], [4, 5, 6]])
print("Tensor 2D:", tensor_2D)
print(tensor_2D.type())

# 用 dtype= 屬性，指定建立的張量資料型態
tensor_1D = torch.tensor([1, 2, 3], dtype=torch.float)
print("Tensor 1D:", tensor_1D)
print(tensor_1D.type())
```

```
Tensor 1D: tensor([1, 2, 3])
torch.LongTensor
Tensor 2D: tensor([[1, 2, 3],
        [4, 5, 6]])
torch.LongTensor

Tensor 1D: tensor([1., 2., 3.])
torch.FloatTensor
```

建立特殊張量

- 建立「零張量」

```
1  import tensor
2
3  zero_1D = torch.zeros((3,))
4  print("Zero 1D:", zero_1D)
5
6  zero_2D = torch.zeros((2, 3))
7  print("Zero 2D:", zero_2D)
```

預設是浮點數

一維零張量 [0. 0. 0.]

二維零張量
[[0. 0. 0.]
 [0. 0. 0.]]

- 建立「單位張量」

```
1  import tensor
2
3  identity_1D = torch.eye(1)
4  print("Identity 1D:", identity_1D)
5
6  identity_2D = torch.eye(2)
7  print("Identity 2D:", identity_2D)
```

預設是浮點數

一維單位張量 [[1.]]

二維單位張量
[[1. 0.]
 [0. 1.]]

建立特殊張量

- 建立全為 1 的「常數張量」

```
1  import tensor
2
3  ones_1D = torch.ones((3,))
4  print("Ones 1D:", ones_1D)
5
6  ones_2D = torch.ones((2, 3))
7  print("Ones 2D:", ones_2D)
```

預設是浮點數

一維常數張量 [1. 1. 1.]

二維常數張量
[[1. 1. 1.]
 [1. 1. 1.]]

- 建立全為特定數字的「常數張量」

```
1  import tensor
2
3  full_1D = torch.full((3,), 7)
4  print("Full 1D:", full_1D)
5
6  full_2D = torch.full((2, 3), 7)
7  print("Full 2D:", full_2D)
```

一維常數張量 [7 7 7]

二維常數張量
[[7 7 7]
 [7 7 7]]

隨堂練習：零張量、單位張量、常數張量

- **用下列程式碼，建立「零張量」，並且印出：**
  - zero_1D = torch.zeros((3,))
  - zero_2D = torch.zeros((2, 3))

- **用下列程式碼，建立「單位張量」，並且印出：**
  - identity_1D = torch.eye(1)
  - identity_2D = torch.eye(2)

- **用下列程式碼，建立「常數張量」，並且印出：**
  - 全為 1
    - ones_1D = torch.ones((3,))
    - ones_2D = torch.ones((2, 3))
  - 全為 7
    - full_1D = torch.full((3,), 7)
    - full _2D = torch.full((2,3), 7)

與其它資料結構轉換

- Python List → Tensors

```python
import tensor

list_1D = [1, 2, 3]
tensor_1D = torch.tensor(list_1D)
print("Tensor 1D:", tensor_1D)
```

Tensor 1D: tensor([1, 2, 3])

- Tensors → Python List

```python
import tensor

list_1D = tensor_1D.tolist()
print("List 1D:", list_1D)
```

List 1D: [1, 2, 3]

與其它資料結構轉換

- NumPy NDArray → Tensors

```python
import tensor
import numpy as np

ndarray_1D = np.array([1, 2, 3])
tensor_1D = torch.from_numpy(ndarray_1D)
print("Tensor 1D:", tensor_1D)
```

Tensor 1D: tensor([1, 2, 3])

- Tensors → NumPy NDArray

```python
import tensor
import numpy as np

ndarray_1D = tensor_1D.numpy()
print("NDArray 1D:", ndarray_1D)
```

NDArray 1D: [1 2 3]

隨堂練習：與其它資料結構轉換

- Python List 與 Tensors 互轉
  - 請輸入下列程式碼，並執行看看。學習如何將資料與 Python List 互轉：

```python
import tensor

list_1D = [1, 2, 3]
tensor_1D = torch.tensor(list_1D)
print("Tensor 1D:", tensor_1D)

list_1D = tensor_1D.tolist()
print("List 1D:", list_1D)
```

- NumPy NDArray 與 Tensors 互轉
  - 請輸入下列程式碼，並執行看看。學習如何將資料與 NumPy NDArray 互轉：

```python
import tensor
import numpy as np

ndarray_1D = np.array([1, 2, 3])
tensor_1D = torch.from_numpy(ndarray_1D)
print("Tensor 1D:", tensor_1D)

ndarray_1D = tensor_1D.numpy()
print("NDArray 1D:", ndarray_1D)
```

將張量在 CPU 與 GPU 間互轉

```python
import tensor

# 建立一個 Tensor
tensor_1D = torch.tensor([1, 2, 3])
print("Tensor 1D:", tensor_1D)
print(tensor_1D.type())

# 將張量由 CPU 轉到 GPU 去
tensor_1D_GPU = tensor_1D.to(device)
print("Tensor 1D GPU:", tensor_1D_GPU)
print(tensor_1D_GPU.type())

# 將張量由 GPU 轉到 CPU 去
tensor_1D_CPU = tensor_1D_GPU.to(torch.device("cpu"))
print("Tensor 1D CPU:", tensor_1D_CPU)
print(tensor_1D_CPU.type())
```

資料正確

Tensor 1D: tensor([1, 2, 3])
torch.LongTensor

目前位於 CPU

Tensor 1D GPU: tensor([1, 2, 3], device='cuda:0')
torch.cuda.LongTensor

目前位於 GPU

Tensor 1D CPU: tensor([1, 2, 3])
torch.LongTensor

又回到 CPU 了

隨堂練習：CPU 與 GPU 互轉

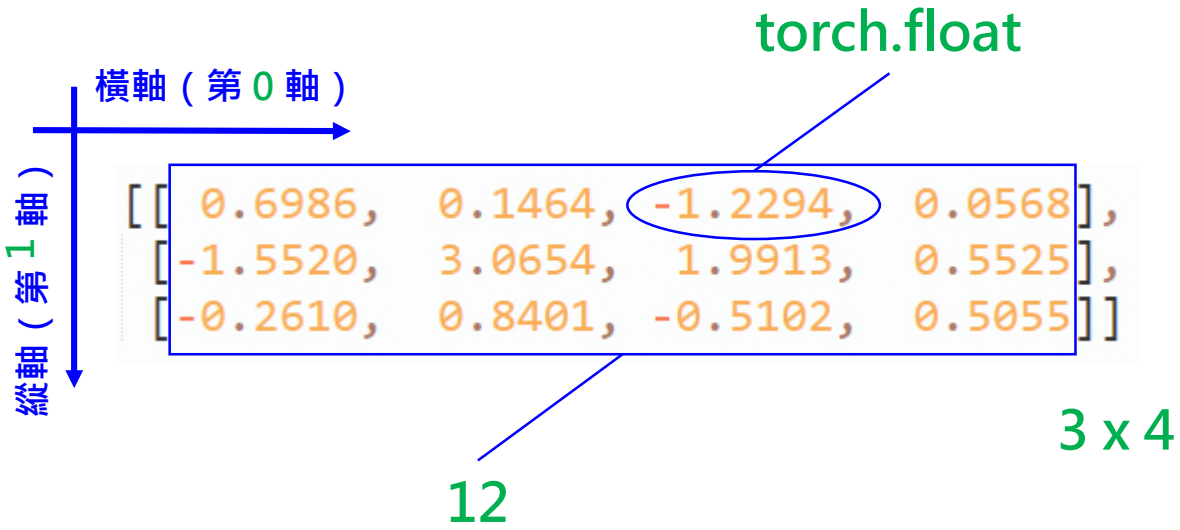
- 請輸入下列程式碼，並執行看看，資料是否在 CPU/GPU 間互轉：

```python
import tensor

# 建立一個 Tensor
tensor_1D = torch.tensor([1, 2, 3])
print("Tensor 1D:", tensor_1D)
print(tensor_1D.type())

# 將張量由 CPU 轉到 GPU 去
tensor_1D_GPU = tensor_1D.to(device)
print("Tensor 1D GPU:", tensor_1D_GPU)
print(tensor_1D_GPU.type())

# 將張量由 GPU 轉到 CPU 去
tensor_1D_CPU = tensor_1D_GPU.to(torch.device("cpu"))
print("Tensor 1D CPU:", tensor_1D_CPU)
print(tensor_1D_CPU.type())
```

```
Tensor 1D: tensor([1, 2, 3])
torch.LongTensor
Tensor 1D GPU: tensor([1, 2, 3], device='cuda:0')
torch.cuda.LongTensor
Tensor 1D CPU: tensor([1, 2, 3])
torch.LongTensor
```

張量本身資訊之取得

張量本身資訊有哪些？



- **軸數（Axes）**
  - 俗稱「維度（Dimension）」
  - 在 PyTorch 內的正式名稱為「軸」。
  - 目前的張量軸數 = 2。
- **維度（Dimension）**
  - 俗稱「行列數」、「形狀」。
  - PyTorch 用 Tuple 來表示之。
  - 目前的維度/形狀 = (3, 4)。
- **元素個數（Number of Elements）**
  - 張量元素的總個數。
  - 目前的元素個數為 3x4 = 12 個。
- **元素資料型態（Data Type, dtype）**
  - 張量內每個元素的資料型態。
  - 目前的元素資料型態為 torch.float。

取得張量本身資訊

```python
import tensor   # torch

# 取得一個張量的「軸數」
tensor_2D = torch.tensor([[1, 2, 3], [4, 5, 6]])
print("Tensor 2D:", tensor_2D)
print("Axes:", tensor_2D.dim())

# 取得一個張量的「維度/形狀」
print("Shape:", tensor_2D.shape)
print("Size:", tensor_2D.size())

# 取得一個張量的「元素數」
print("Number of Elements:", tensor_2D.numel())

# 取得一個張量的「資料型態」
print("Data Type:", tensor_2D.dtype)
```

```
Tensor 2D: tensor([[1, 2, 3],
       [4, 5, 6]])
Axes: 2
```

```
Shape: torch.Size([2, 3])
Size: torch.Size([2, 3])
```

```
Number of Elements: 6
```

```
Data Type: torch.int64
```

隨堂練習：取得張量本身資訊

- 請輸入下列程式碼，並執行看看，是否能取得張量本身資訊：

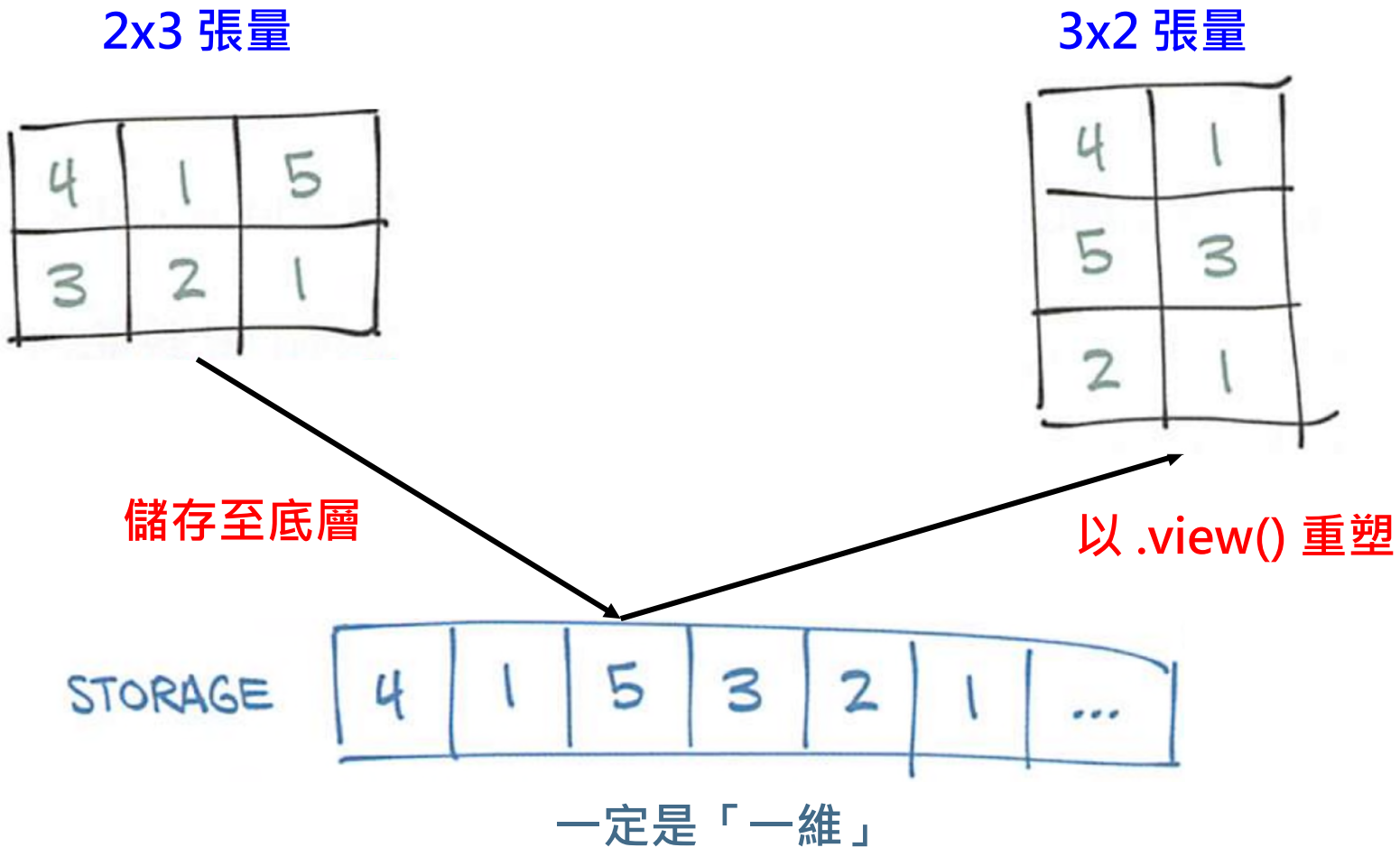
```python
import tensor torch

# 取得一個張量的「軸數」
tensor_2D = torch.tensor([[1, 2, 3], [4, 5, 6]])
print("Tensor 2D:", tensor_2D)
print("Axes:", tensor_2D.dim())

# 取得一個張量的「維度/形狀」
print("Shape:", tensor_2D.shape)
print("Size:", tensor_2D.size())

# 取得一個張量的「元素數」
print("Number of Elements:", tensor_2D.numel())

# 取得一個張量的「資料型態」
print("Data Type:", tensor_2D.dtype)
```

```
Tensor 2D: tensor([[1, 2, 3],
        [4, 5, 6]])
Axes: 2
Shape: torch.Size([2, 3])
Size: torch.Size([2, 3])
Number of Elements: 6
Data Type: torch.int64
```

張量元素的讀寫

張量如何儲存

- 原理


2x3 張量
4 1 5
3 2 1
3x2 張量
4 1
5 3
2 1
儲存至底層
以 .view() 重塑
STORAGE
4 1 5 3 2 1 ...
一定是「一維」

張量如何儲存

- 程式碼

```python
import torch

# 建造一個 2x3 的張量
tensor_2D = torch.tensor([[4.0, 1.0, 5.0], [3.0, 2.0, 1.0]])
print("Tensor 2D:", tensor_2D)

# 將張量的 storage 印出
print("Storage:", tensor_2D.storage())

# 以 view 重塑為 3x2 張量
tensor_3x2 = tensor_2D.view(3, 2)
print("Tensor 3x2:", tensor_3x2)
```

```
Tensor 2D: tensor([[4., 1., 5.],
        [3., 2., 1.]])

Storage:  4.0
 1.0
 5.0
 3.0
 2.0
 1.0

Tensor 3x2: tensor([[4., 1.],
        [5., 3.],
        [2., 1.]])
```

隨堂練習：張量儲存原理

- 請輸入下列程式碼，並執行看看，並透過結果理解張量儲存原理：

```python
import torch

# 建造一個 2x3 的張量
tensor_2D = torch.tensor([[4.0, 1.0, 5.0], [3.0, 2.0, 1.0]])
print("Tensor 2D:", tensor_2D)

# 將張量的 storage 印出
print("Storage:", tensor_2D.storage())

# 以 view 重塑為 3x2 張量
tensor_3x2 = tensor_2D.view(3, 2)
print("Tensor 3x2:", tensor_3x2)
```

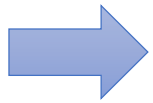
```
Tensor 2D: tensor([[4., 1., 5.],
        [3., 2., 1.]])
Storage:  4.0
 1.0
 5.0
 3.0
 2.0
 1.0
[torch.storage.TypedStorage(dtype=torch.float32, device=cpu) of size 6]
Tensor 3x2: tensor([[4., 1.],
        [5., 3.],
        [2., 1.]])
```

張量元素讀寫

- **讀取**張量元素

```
[[4.0, 1.0, 5.0],
 [3.0, 2.0, 1.0]]
```

```
1  import torch
2
3  tensor_2D = torch.tensor([[4.0, 1.0, 5.0], [3.0, 2.0, 1.0]])
4  print("Element at (1, 2):", tensor_2D[1, 2])
```

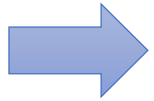


```
1.0
```

- **寫入**張量元素

```
9.0
[[4.0, 1.0, 5.0],
 [3.0, 2.0, 1.0]]
```

```
1  import torch
2
3  tensor_2D = torch.tensor([[4.0, 1.0, 5.0], [3.0, 2.0, 1.0]])
4  tensor_2D[0, 0] = 9.0
5  print("Tensor 2D:", tensor_2D)
```



```
[[9.0  1.0  5.0]
 [3.0  2.0  1.0]]
```

隨堂練習：讀寫張量元素

- 請輸入下列程式碼，並執行看看，是否能夠讀寫心目中的張量元素：

```python
import torch

# 建造一個 2x3 的張量
tensor_2D = torch.tensor([[4.0, 1.0, 5.0], [3.0, 2.0, 1.0]])
print("Tensor 2D:", tensor_2D)

# 讀取位於索引值 (1, 2) 的元素
print("Element at (1, 2):", tensor_2D[1, 2])

# 將索引值 (0, 0) 的元素寫入為 9.0
tensor_2D[0, 0] = 9.0
print("Tensor 2D:", tensor_2D)
```

```
Tensor 2D: tensor([[4., 1., 5.],
        [3., 2., 1.]])
Element at (1, 2): tensor(1.)
Tensor 2D: tensor([[9., 1., 5.],
        [3., 2., 1.]])
```

張量樣本點的產生

線性樣本點

- 產生「線性」樣本點

```
1  import torch
2
3  sample_1D = torch.arange(0.0, 5.0, 0.2)
4  print("Sample 1D:", sample_1D)
```

arange = Array RANGE = [0.0, 5.0) 浮點數序列

```
[0.  0.2 0.4 0.6 0.8
 1.  1.2 1.4 1.6 1.8
 2.  2.2 2.4 2.6 2.8
 3.  3.2 3.4 3.6 3.8
 4.  4.2 4.4 4.6 4.8]
```

- 產生「線性」樣本點 + 洗牌

```
1  import torch
2
3  # 第 0 軸元素個數
4  index_num = sample_1D.size(0)
5
6  # 產生 index_num 個索引值亂數
7  index_rand = torch.randperm(index_num)
8
9  sample_1D = sample_1D[index_rand]
10 print("Sample 1D:", sample_1D)
```

傳回 25

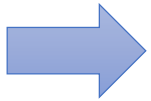
```
[14, 22, 15, 11,  0,
 23,  4,  5,  1, 10,
  9,  7,  8,  2,  6,
 12, 24, 17, 16, 19,
 18, 20,  3, 13, 21]
```

```
[2.8, 4.4, 3.0, 2.2, 0.0,
 4.6, 0.8, 1.0, 0.2, 2.0,
 1.8, 1.4, 1.6, 0.4, 1.2,
 2.4, 4.8, 3.4, 3.2, 3.8,
 3.6, 4.0, 0.6, 2.6, 4.2]
```

線性樣本點

- 產生「線性」樣本點 + 更改維度

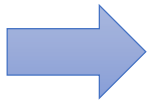
```
1  import torch
2
3  sample_1D = torch.arange(0.0, 5.0, 0.2)
4  sample_2D = sample_1D.view(5, 5)
5  print("Sample 2D:", sample_2D)
```



```
[[0.  0.2 0.4 0.6 0.8]
 [1.  1.2 1.4 1.6 1.8]
 [2.  2.2 2.4 2.6 2.8]
 [3.  3.2 3.4 3.6 3.8]
 [4.  4.2 4.4 4.6 4.8]]
```

- 產生「線性」樣本點 + 更改資料型態

```
1  import torch
2
3  sample_1D = torch.arange(0.0, 5.0, 0.2)
4  sample_1D = sample_1D.type(torch.int32)
5  print("Sample 2D:", sample_1D)
```



無條件捨去至整數

```
[0, 0, 0, 0, 0,
 1, 1, 1, 1, 1,
 2, 2, 2, 2, 2,
 3, 3, 3, 3, 3,
 4, 4, 4, 4, 4]
```

隨堂練習：線性樣本點

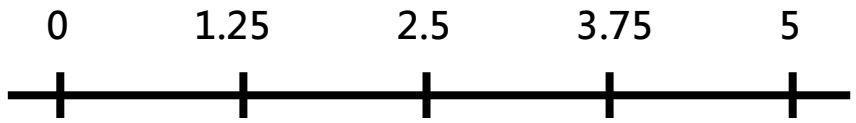
- 請先用 arange() 建立「線性樣本點」張量，並將它印出
  - sample_1D = torch.arange(0., 5., 0.2)
- 用下列程式碼「弄亂」它，並將結果印出
  - index_num = sample_1D.size(0)
  - index_rand = torch.randperm(index_num)
  - sample_1D = sample_1D[index_rand]
- 用下列程式碼，將維度從 1x25 改成 5x5，並將之印出
  - sample_1D = sample_1D.view(5, 5)
- 用下列程式碼，將元素型態改為「整數」，並將之印出
  - sample_1D = sample_1D.type(torch.int32)
- 參考程式碼如右所示：

```
1  import torch
2
3  sample_1D = torch.arange(0.0, 5.0, 0.2)
4  print("Sample 1D:", sample_1D)
5
6  index_num = sample_1D.size(0)
7  index_rand = torch.randperm(index_num)
8  sample_1D = sample_1D[index_rand]
9  print("Sample 1D:", sample_1D)
10
11 sample_2D = sample_1D.view(5, 5)
12 print("Sample 2D:", sample_2D)
13
14 sample_2D = sample_1D.view(5, 5)
15 print("Sample 2D:", sample_2D)
16
17 sample_1D = sample_1D.type(torch.int32)
18 print("Sample 2D:", sample_1D)
```

等差 / 等比樣本點

- 產生「等差」樣本點

```
1  import torch
2
3  sample_1D = torch.linspace(0.0, 5.0, 5)
4  print("Sample 1D:", sample_1D)
```

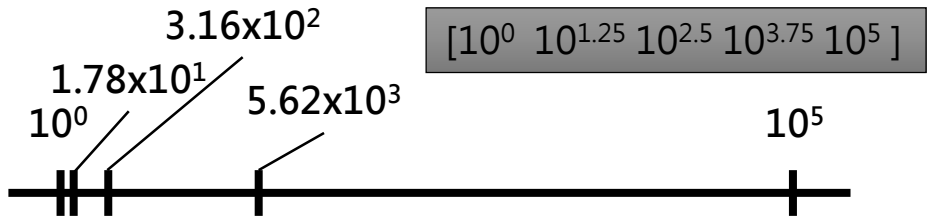


[0.0 1.25 2.5 3.75 5.0]

- 產生「等比」樣本點

更有效率！

```
1  import torch
2
3  sample_1D = torch.logspace(0.0, 5.0, 5)
4  print("Sample 1D:", sample_1D)
```



[1.0000e+00, 1.7783e+01, 3.1623e+02, 5.6234e+03, 1.0000e+05]

隨堂練習：等差 / 等比樣本點

- 請用下列這幾道指令，建立「等差」的樣本點：

```
1  sample_1D = torch.linspace(0.0, 5.0, 5)
2  print("Sample 1D:", sample_1D)
```

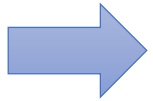
- 再用下列這幾道指令，建立「等比」的樣本點：

```
1  sample_1D = torch.logspace(0.0, 5.0, 5)
2  print("Sample 1D:", sample_1D)
```

一般亂數樣本點

- 產生整數「亂數」樣本點

```
1  import torch
2
3  sample_1D = torch.randint(1, 7, size=(15,))
4  print("Sample 1D:", sample_1D)
```

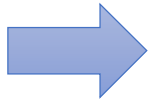


[4 6 4 6 5 1 6 5 4 1 2 1 5 5 3]

產生 [1, 7) 之間的 15x1 個樣本點 = 模擬擲骰子 15 次

- 產生浮點數「亂數」樣本點

```
1  import torch
2
3  sample_2D = torch.rand(2, 3)
4  print("Sample 2D:", sample_2D)
```



[[0.7515, 0.8034, 0.8802],
 [0.6487, 0.7711, 0.3203]]

產生 [0, 1) 之間的浮點亂數 2x3 個
(其它範圍亂數：以 r [0, 1) + b 製作之)

隨堂練習：一般亂數樣本點

- 請用下列方法，模擬出擲 15 次骰子的結果，並將之印出
  - sample_1D = torch.randint(1, 7, size=(15,))
- 請用下列方法，製造出 2x3 個 0~1 之間的浮點數，並將之印出
  - sample_2D = torch.rand(2, 3)
- 參考程式碼如下所示

```
1  import torch
2
3  sample_1D = torch.randint(1, 7, size=(15,))
4  print("Sample 1D:", sample_1D)
5
6  sample_2D = torch.rand(2, 3)
7  print("Sample 2D:", sample_2D)
```

常態亂數樣本點

- 產生「標準常態分佈」（平均值 = 0，標準差 = 1）

```
1  import torch
2
3  sample_2D = torch.randn(3, 5)
4  print("Sample 2D:", sample_2D)
```

```
[[-4.8321e-01,  8.5426e-01, -2.5195e-01,  2.9390e-04,  8.0159e-01],
 [-2.6223e-01,  2.6911e-02, -2.6561e+00,  9.8266e-01, -4.4369e-01],
 [ 2.9124e-01, -1.5893e+00,  8.0097e-01,  4.8534e-01,  1.2237e+00]]
```

產生 3x5 個符合「標準常態分佈」的樣本點

- 產生「一般常態分佈」

```
1  import torch
2
3  sample_2D = torch.normal(10.0, 2.0, size=(3, 5))
4  print("Sample 2D:", sample_2D)
```

```
[[ 9.4878, 11.0112,  9.0874,  8.9270, 10.5239],
 [ 8.8640,  6.3615,  9.6121,  7.0035,  8.2266],
 [11.6888, 14.0326,  7.4238, 15.1021,  6.3668]]
```

產生 3x5 個符合「平均值 = 10, 標準差 =2」的樣本點

隨堂練習：常態亂數樣本點

- 請輸入下列程式碼，產生 3x5 個「標準常態分佈」樣本點
  - sample_2D = torch.randn(3, 5)

- 請輸入下列程式碼，產生 3x5 個「一般常態分佈」樣本點
  - sample_2D = torch.normal(10.0, 2.0, size=(3, 5))

- 參考程式碼如下所示：

```
1  import torch
2
3  sample_2D = torch.randn(3, 5)
4  print("Sample 2D:", sample_2D)
5
6  sample_2D = torch.normal(10.0, 2.0, size=(3, 5))
7  print("Sample 2D:", sample_2D)
```

張量基本運算

張量運算：負數&加減

```python
import torch

X1 = torch.tensor([1, 2, 3])
X2 = torch.tensor([20, 36, 40])

print(torch.neg(X1))
print(torch.add(X1, X2))
print(torch.sub(X1, X2))
```

1 $X1 = [a_{00} \quad a_{01} \quad a_{02}]$
$= [1 \quad 2 \quad 3]$

$X2 = [b_{00} \quad b_{01} \quad b_{02}]$
$= [20 \quad 36 \quad 40]$

2 .neg (X1)：與 -X1 效果相同
$-X1 = [-a_{00} \quad -a_{01} \quad -a_{02}]$
$= [-1 \quad -2 \quad -3]$

3 .add(X1, X2) ：與 X1+X2 效果相同
$X1 + X2 = [a_{00} + b_{00} \quad a_{01} + b_{01} \quad a_{02} + b_{02}]$
$= [1 + 20 \quad 2 + 36 \quad 3 + 40]$
$= [21 \quad 38 \quad 43]$

4 .sub (X1, X2) ：與 X1-X2 效果相同
$X1 - X2 = [a_{00} - b_{00} \quad a_{01} - b_{01} \quad a_{02} - b_{02}]$
$= [1 - 20 \quad 2 - 36 \quad 3 - 40]$
$= [-19 \quad -34 \quad -37]$

隨堂練習：張量「負數」&「加減」

- **請先定義如下的張量：**
  - X1 = torch.**tensor**([1, 2, 3])
  - X2 = torch.**tensor**([20, 36, 40])

- **用下列指令，練習張量「負數」與「加減」運算：**
  - torch.**neg** (X1)
  - torch.**add**(X1, X2)
  - torch.**sub** (X1, X2)

- **參考程式碼如右圖所示：**

```
1  import torch
2
3  X1 = torch.tensor([1, 2, 3])
4  X2 = torch.tensor([20, 36, 40])
5
6  print(torch.neg(X1))
7  print(torch.add(X1, X2))
8  print(torch.sub(X1, X2))
```

張量運算：各種乘積

```python
import torch

X1 = torch.tensor([1, 2, 3])
X2 = torch.tensor([20, 36, 40])

print(torch.mul(X1, X2))
print(torch.matmul(X1, X2))
print(torch.dot(X1, X2))
print(torch.inner(X1, X2))
print(torch.cross(X1, X2))
print(torch.outer(X1, X2))
```

1 .mul (X1, X2)：係數積

$$X1 * X2 = [a_{00} * b_{00} \quad a_{01} * b_{01} \quad a_{02} * b_{02}]$$
$$= [1 * 20 \quad 2 * 36 \quad 3 * 40]$$
$$= [20 \quad 72 \quad 120]$$

2 .matmul(X1, X2)：一般乘積（需遵守 $m \times p \ * p \times n$）

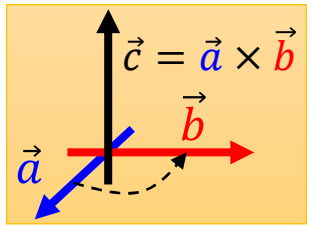
$$\overrightarrow{X1} * \overrightarrow{X2} = a_{00} * b_{00} + a_{01} * b_{01} + a_{02} * b_{02}$$
$$= 1 * 20 + 2 * 36 + 3 * 40$$
$$= 212$$

3 .dot(X1, X2)：點積，適用於一維向量相乘結果

$$\overrightarrow{X1} \cdot \overrightarrow{X2} = a_{00} * b_{00} + a_{01} * b_{01} + a_{02} * b_{02}$$
$$= 1 * 20 + 2 * 36 + 3 * 40$$
$$= 212$$

4 .inner(X1, X2)：內積，與 dot() 類似，適用於高維向量

$$[X1] \odot [X2] = [a_{00} * b_{00} + a_{01} * b_{01} + a_{02} * b_{02}]$$
$$= [1 * 20 + 2 * 36 + 3 * 40]$$
$$= 212$$



5 .cross(X1, X2)：叉積（可求得兩向量之「法向量」）

$$\overrightarrow{X1} \times \overrightarrow{X2} = \begin{bmatrix} i & j & k \\ a_{00} & a_{01} & a_{02} \\ b_{00} & b_{01} & b_{02} \end{bmatrix} = \begin{bmatrix} a_{01} & a_{02} \\ b_{01} & b_{02} \end{bmatrix} i - \begin{bmatrix} a_{00} & a_{02} \\ b_{00} & b_{02} \end{bmatrix} j + \begin{bmatrix} a_{00} & a_{01} \\ b_{00} & b_{01} \end{bmatrix} k$$
$$\vec{C} = [i \quad j \quad k] = [a_{01}b_{02} - a_{02}b_{01} \quad a_{02}b_{00} - a_{00}b_{02} \quad a_{00}b_{01} - a_{01}b_{00}]$$

6 .outer(X1, X2)：外積、倍數積（結果為「矩陣」）

$$[X1] \otimes [X2] = \begin{bmatrix} a_{00} \\ a_{01} \\ a_{02} \end{bmatrix} [b_{00} \quad b_{01} \quad b_{02}] = \begin{bmatrix} a_{00}b_{00} & a_{00}b_{01} & a_{00}b_{02} \\ a_{01}b_{00} & a_{01}b_{01} & a_{01}b_{02} \\ a_{02}b_{00} & a_{02}b_{01} & a_{02}b_{02} \end{bmatrix}$$

隨堂練習：各種張量乘積

- 請先定義如下的張量：
  - X1 = torch.tensor([1, 2, 3])
  - X2 = torch.tensor([20, 36, 40])

- 用下列指令，練習張量「係數積、乘積、點積、內積、叉積、外積」運算：
  - 係數積：torch.mul (X1, X2)
  - 乘積：torch.matmul (X1, X2)
  - 點積：torch.dot(X1, X2)
  - 內積：torch.inner(X1, X2)
  - 叉積：torch.cross(X1, X2)
  - 外積：torch.outer(X1, X2)

- 參考程式碼如右圖所示：

```
1  import torch
2
3  X1 = torch.tensor([1, 2, 3])
4  X2 = torch.tensor([20, 36, 40])
5
6  print(torch.mul(X1, X2))
7  print(torch.matmul(X1, X2))
8  print(torch.dot(X1, X2))
9  print(torch.inner(X1, X2))
10 print(torch.cross(X1, X2))
11 print(torch.outer(X1, X2))
```

張量運算：除法相關運算

```python
import torch

X1 = torch.tensor([1, 2, 3])
X2 = torch.tensor([20, 36, 40])

print(torch.reciprocal(X1))
print(torch.div(X1, X2))
print(torch.remainder(X1, X2))

X = torch.Tensor([[1, 2], [3, 4]])
Y = torch.inverse(X)
print(torch.matmul(X, Y))
```

1. .reciprocal(X1)：取倒數

$$[1] / [X1] = \left[\frac{1}{a_{00}} \quad \frac{1}{a_{01}} \quad \frac{1}{a_{02}}\right]$$
$$= \left[\frac{1}{1} \quad \frac{1}{2} \quad \frac{1}{3}\right]$$
$$= [1.0000 \quad 0.5000 \quad 0.3333]$$

2. .div(X1, X2)：係數除法（同 X1/X2）

$$[X1] / [X2] = \left[\frac{a_{00}}{b_{00}} \quad \frac{a_{01}}{b_{01}} \quad \frac{a_{02}}{b_{02}}\right]$$
$$= \left[\frac{1}{20} \quad \frac{2}{36} \quad \frac{3}{40}\right]$$
$$= [0.0500 \quad 0.0556 \quad 0.0750]$$

3. .remainder(X1, X2)：取餘數（同 X1%X2）

$$[X1]\%[X2] = \left[\frac{a_{00}}{b_{00}} \quad \frac{a_{01}}{b_{01}} \quad \frac{a_{02}}{b_{02}}\right]$$
$$= \left[\frac{1}{20} \quad \frac{2}{36} \quad \frac{3}{40}\right]$$
$$= [1 \quad 2 \quad 3]$$

4. .inverse(X)：矩陣除法（反矩陣，需用浮點數）

$$[X] \div [Y] = [X] \times [Y]^{-1}$$
$$= \begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} \times \begin{bmatrix}-2 & 1\\ 1.5 & -0.5\end{bmatrix}$$
$$= \begin{bmatrix}1.0 & 0.0\\ 0.0 & 1.0\end{bmatrix}$$

隨堂練習：各種除法相關運算

- 請先定義如下的張量：
  - X1 = torch.tensor([1, 2, 3])
  - X2 = torch.tensor([20, 36, 40])
  - X = torch.tensor([[1, 2], [3, 4]])

Tensor

- 用下列指令，練習張量「倒數、係數除法、取餘數、張量除法」運算：
  - 倒數：torch.reciprocal(X1)
  - 係數除法：torch.div(X1, X2)
  - 取餘數：torch.remainder(X1, X2)
  - 張量除法（反矩陣）：torch.inverse(X)

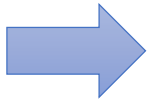
- 參考程式碼如右圖所示：

```
1  import torch
2
3  X1 = torch.tensor([1, 2, 3])
4  X2 = torch.tensor([20, 36, 40])
5
6  print(torch.reciprocal(X1))
7  print(torch.div(X1, X2))
8  print(torch.remainder(X1, X2))
9
10 X = torch.Tensor([[1, 2], [3, 4]])
11 Y = torch.inverse(X)
12 print(torch.matmul(X, Y))
```

張量切片運算（Slicing）

- 一維張量切片運算

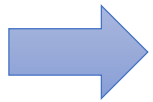
```
1  import torch
2
3  slice_1D = torch.randint(20, size=(20,))
4
5  print(slice_1D)
6  print(slice_1D[:5])
```



```
[ 5 14 17 18 14  0  3  6 12  1 14  3  6 14 17  9 11  0  3 18]
```

- 二維張量切片運算

```
1  import torch
2
3  slice_2D = torch.randint(20, size=(5, 5))
4
5  print(slice_2D)
6  print(slice_2D[:3, :3])
```



```
[[ 9 15  3 13 15]
 [ 7 11 10  3  0]
 [14 13 17  3 10]
 [ 9  2 13 18 17]
 [14 10  7  6 15]]
```

隨堂練習：張量切片運算（Slicing）

- 請依照前一頁投影片的內容，練習一維、二維張量的切片運算。

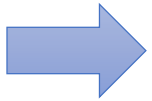
- 參考程式碼如下所示：

```
1  import torch
2
3  slice_1D = torch.randint(20, size=(20,))
4  print(slice_1D)
5  print(slice_1D[:5])
6
7  slice_2D = torch.randint(20, size=(5, 5))
8  print(slice_2D)
9  print(slice_2D[:3, :3])
```

維度壓縮與擴張

- 維度壓縮（Squeeze）：去除不必要的維度（如：1x3 → 3）

```
1  import torch
2
3  sample_2D = torch.tensor([[1, 2, 3]])
4
5  print("Dimension:", sample_2D.size())
6  print(sample_2D)
7
8  sample_1D = sample_2D.squeeze()
9  print("Dimension:", sample_1D.size())
10 print(sample_1D)
```



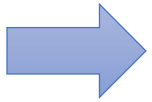
[[1 2 3]] 2 維（1x3）

[1 2 3] 1 維（3）

- 維度擴張（Unsqueeze）：增加所需的的維度（如：3 → 1x3）

```
1  import torch
2
3  sample_1D = torch.tensor([1, 2, 3])
4
5  print("Dimension:", sample_1D.size())
6  print(sample_1D)
7
8  sample_2D = sample_1D.unsqueeze(0)
9  print("Dimension:", sample_2D.size())
10 print(sample_2D)
```

擴張的維度位於第 0 軸



[1 2 3] 1 維（3）

[[1 2 3]] 2 維（1x3）

隨堂練習：維度壓縮與擴張

- 請依照前一頁投影片的內容，練習張量維度的壓縮與擴張。
- 參考程式碼如下所示：

```python
# 維度壓縮
sample_2D = torch.tensor([[1, 2, 3]])

print("Dimension:", sample_2D.size())
print(sample_2D)

sample_1D = sample_2D.squeeze()
print("Dimension:", sample_1D.size())
print(sample_1D)
# ---------------------------
# 維度擴張
sample_1D = torch.tensor([1, 2, 3])

print("Dimension:", sample_1D.size())
print(sample_1D)

sample_2D = sample_1D.unsqueeze(0)
print("Dimension:", sample_2D.size())
print(sample_2D)
```

```
Dimension: torch.Size([1, 3])
tensor([[1, 2, 3]])
Dimension: torch.Size([3])
tensor([1, 2, 3])
```

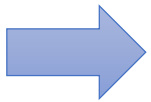
```
Dimension: torch.Size([3])
tensor([1, 2, 3])
Dimension: torch.Size([1, 3])
tensor([[1, 2, 3]])
```

張量合併：cat()

cat = conCATenation

- 以列為邊界合併

```
1  import torch
2
3  X1 = torch.tensor([[1, 2], [3, 4]])
4  X2 = torch.tensor([[5, 6], [7, 8]])
5
6  print(torch.cat((X1, X2), dim=0))
```

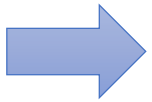


X1 =
[[1, 2],
[3, 4],

X2 =
[5, 6],
[7, 8]]

- 以行為邊界合併

```
1  import torch
2
3  X1 = torch.tensor([[1, 2], [3, 4]])
4  X2 = torch.tensor([[5, 6], [7, 8]])
5
6  print(torch.cat((X1, X2), dim=1))
```



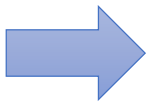
X1 = X2 =

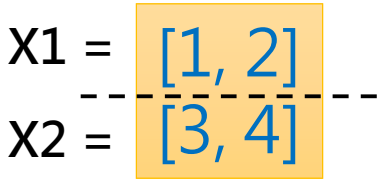
[[1, 2, 5, 6],
[3, 4, 7, 8]]

張量分割：unbind()

- 以列為邊界分割

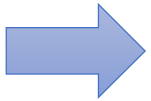
```
1  import torch
2
3  X = torch.tensor([[1, 2], [3, 4]])
4  print(torch.unbind(X, dim=0))
```

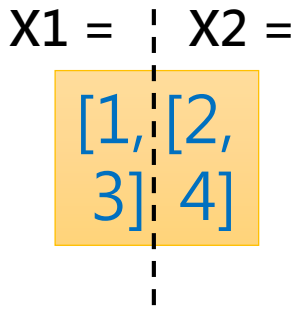




- 以行為邊界分割

```
1  import torch
2
3  X = torch.tensor([[1, 2], [3, 4]])
4  print(torch.unbind(X, dim=1))
```





隨堂練習：張量合併與分割

- 請依照前一頁投影片的內容，練習張量的合併與分割。
- 參考程式碼如下所示：

```
1  # 張量合併
2  X1 = torch.tensor([[1, 2], [3, 4]])
3  X2 = torch.tensor([[5, 6], [7, 8]])
4
5  # 以列為邊界
6  print(torch.cat((X1, X2), dim=0))
7  # 以行為邊界
8  print(torch.cat((X1, X2), dim=1))
9  ----------------------------------
10 # 張量分割
11 X = torch.tensor([[1, 2], [3, 4]])
12
13 # 以列為邊界
14 print(torch.unbind(X, dim=0))
15 # 以行為邊界
16 print(torch.unbind(X, dim=1))
```

```
tensor([[1, 2],
        [3, 4],
        [5, 6],
        [7, 8]])
tensor([[1, 2, 5, 6],
        [3, 4, 7, 8]])
```

```
(tensor([1, 2]), tensor([3, 4]))
(tensor([1, 3]), tensor([2, 4]))
```

張量運算：轉置（Transpose）

```python
import torch

X1 = torch.tensor([1, 2, 3])
X2 = torch.tensor([20, 36, 40])

features = torch.cat((X1, X2)).view(2, 3).T
print(features)
```

features = torch.cat((X1, X2)).view(2, 3).T

$$([1 \quad 2 \quad 3] \quad [20 \quad 36 \quad 40]) \quad \begin{bmatrix} 1 & 2 & 3 \\ 20 & 36 & 40 \end{bmatrix} \quad \begin{bmatrix} 1 & 20 \\ 2 & 36 \\ 3 & 40 \end{bmatrix}$$

一定要用 Tuple 包住再丟入

隨堂練習：轉置（Transpose）

- 請先定義如下的張量：
  - X1 = torch.tensor([1, 2, 3])
  - X2 = torch.tensor([20, 36, 40])

- 用下列指令，練習張量「矩陣轉置」運算：
  - 張量接合： .cat((X1, X2))
  - 維度變更：.view(2, 3)
  - 矩陣轉置：.T

- 參考程式碼如下圖所示：

```
1  import torch
2
3  X1 = torch.tensor([1, 2, 3])
4  X2 = torch.tensor([20, 36, 40])
5
6  features = torch.cat((X1, X2)).view(2, 3).T
7  print(features)
```

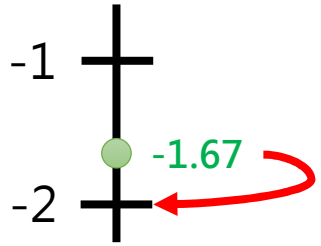
常用函數：小數捨入

num1 = torch.**tensor**([-1.67, -1.01, -0.35, 0.97, 1.63])

torch.**round**(num1, decimals=1)

[-1.7 -1.0 -0.4 1.0 1.6]

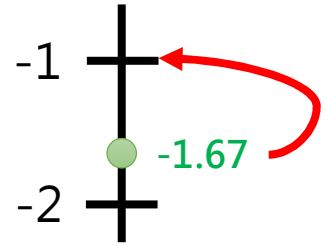
decimal=1 ： 小數點以下一位



torch.**floor**(num1)

[-2.0 -2.0 -1.0 0.0 1.0]

floor = 取「地板」



torch.**ceil**(num1)

[-1.0 -1.0 -0.0 1.0 2.0]

ceil = 取「天花板」

隨堂練習：小數捨入函數

- 請先建立下列張量，並將之印出：
  - num1 = torch.tensor([-1.67, -1.01, -0.35, 0.97, 1.63])

- 練習四捨五入、取地板、取天花板的小數捨入函數：
  - print(torch.**round**(num1, decimals=1))
  - print(torch.**floor**(num1))
  - print(torch.**ceil**(num1))

- 參考原始碼如下所示：

```
1  num1 = torch.tensor([-1.67, -1.01, -0.35, 0.97, 1.63])
2  print(num1)
3
4  print(torch.round(num1, decimals=1))
5  print(torch.floor(num1))
6  print(torch.ceil(num1))
```

常用函數：指數相關函數

```python
import torch

X1 = torch.arange(1.0, 10.0, 1.0).view(3, 3)
print(X1)

print(torch.pow(X1, X1))
print(torch.exp2(X1))
print(torch.exp(X1))
print(torch.sqrt(X1))
```

原始張量

```
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
```

$X1^{X1}$

```
[[1.0000e+00, 4.0000e+00, 2.7000e+01],
 [2.5600e+02, 3.1250e+03, 4.6656e+04],
 [8.2354e+05, 1.6777e+07, 3.8742e+08]]
```

9^9

2^{X1}

```
[[  2.   4.   8.]
 [ 16.  32.  64.]
 [128. 256. 512.]]
```

2^9

e^{x1}

```
[[2.7183e+00, 7.3891e+00, 2.0086e+01],
 [5.4598e+01, 1.4841e+02, 4.0343e+02],
 [1.0966e+03, 2.9810e+03, 8.1031e+03]]
```

e^9

$\sqrt{x1}$

```
[[1.0000, 1.4142, 1.7321],
 [2.0000, 2.2361, 2.4495],
 [2.6458, 2.8284, 3.0000]]
```

$\sqrt{9}$

隨堂練習：指數相關函數

- **請先建立下列張量，並將之印出：**
  - X1 = torch.**arange**(1.0, 10.0, 1.0).**view**(3, 3)

- **用下列指令，練習各種指數相關函數，並將結果印出：**
  - 次方：torch.**pow**(X1, X1)
  - 以 2 為底： torch.**exp2**(X1)
  - 自然指數： torch.**exp**(X1)
  - 開根號： torch.**sqrt**(X1)

- **參考程式碼如下所示：**

```
1  import torch
2
3  X1 = torch.arange(1.0, 10.0, 1.0).view(3, 3)
4  print(X1)
5
6  print(torch.pow(X1, X1))
7  print(torch.exp2(X1))
8  print(torch.exp(X1))
9  print(torch.sqrt(X1))
```

常用函數：對數相關函數

```
1  import torch
2
3  X1 = torch.arange(2, 20, 2).view(3, 3)
4  print(X1)
5
6  print(torch.log10(X1))
7  print(torch.log2(X1))
8  print(torch.log(X1))
```

原始張量

[[2, 4, 6],
 [8, 10, 12],
 [14, 16, 18]]

$\log_{10}X1$

[[0.3010, 0.6021, 0.7782],
 [0.9031, 1.0000, 1.0792],
 [1.1461, 1.2041, 1.2553]]

$\log_{10}18$

$\log_2X1$

[[1.0000, 2.0000, 2.5850],
 [3.0000, 3.3219, 3.5850],
 [3.8074, 4.0000, 4.1699]]

$\log_2 18$

$\log_eX1 = \ln(X1)$

[[0.6931, 1.3863, 1.7918],
 [2.0794, 2.3026, 2.4849],
 [2.6391, 2.7726, 2.8904]]

$\log 18$

隨堂練習：對數相關函數

- **請先建立下列張量，並將之印出：**
  - X1 = torch.arange(2, 20, 2).view(3, 3)

- **用下列指令，練習各種指數相關函數，並將結果印出：**
  - 以 10 為底的對數：torch.log10(X1)
  - 以 2 為底的對數： torch.log2(X1)
  - 自然對數： torch.log(X1)

- **參考程式碼如下所示：**

```
1  import torch
2
3  X1 = torch.arange(2, 20, 2).view(3, 3)
4  print(X1)
5
6  print(torch.log10(X1))
7  print(torch.log2(X1))
8  print(torch.log(X1))
```

常用函數：三角函數

```python
import torch

deg = torch.tensor([0, 30, 60, 90])
rad = torch.deg2rad(deg)
print(torch.rad2deg(rad))

print(torch.sin(rad))
print(torch.cos(rad))
print(torch.tan(rad))
```

1

弧度		弧度
0		0
30	.deg2rad() →	π/6
60	← .rad2deg()	π/3
90		π/2

角度		0	π/6 (30)	π/3 (60)	π/2 (90)
2 sin()	理論值	0	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	1
	實際值	0.0	0.5	0.8660254	1.0
3 cos()	理論值	1	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	0
	實際值	1.0	0.8660254	0.5	0.0
4 tan()	理論值	0	$\frac{\sqrt{3}}{3}$	$\sqrt{3}$	$\pm\infty$
	實際值	0.0	0.5773503	1.7320508	1.63312394e+16

隨堂練習：三角函數

- 請先建立下列張量：
  - deg = torch.tensor([0, 30, 60, 90])
- 請用 deg2rad()，將弧度換成弧度，再用 rad2deg() 換回來驗證：
  - rad = torch.deg2rad(deg)
  - print(torch.rad2deg(rad))
- 用下列三個函數，練習三角函數的用法：
  - torch.sin(rad)
  - torch.cos(rad)
  - torch.tan(rad)
- 參考程式碼如右圖所示：

```
tensor([ 0.0000, 30.0000, 60.0000, 90.0000])
tensor([0.0000, 0.5000, 0.8660, 1.0000])
tensor([ 1.0000e+00,  8.6603e-01,  5.0000e-01, -4.3711e-08])
tensor([ 0.0000e+00,  5.7735e-01,  1.7321e+00, -2.2877e+07])
```

```python
import torch

deg = torch.tensor([0, 30, 60, 90])
rad = torch.deg2rad(deg)
print(torch.rad2deg(rad))

print(torch.sin(rad))
print(torch.cos(rad))
print(torch.tan(rad))
```

統計量的計算

```
1  import torch
2
3  stat1 = torch.arange(1.0, 11.0)
4  print(stat1)
5
6  print(stat1.min())
7  print(stat1.max())
8  print(stat1.sum())
9  print(stat1.mean())
10 print(stat1.median())
11 print(stat1.std())
```

```
[1., 2., 3., 4., 5., 6., 7., 8., 9., 10.]
1.
10.
55.
5.5000
5.
3.0277
```

1. 原始資料
2. 取張量中最小元素
3. 取張量中最大元素
4. 取張量元素總和
5. 取張量元素平均
6. 取張量元素中位數
7. 取張量元素標準差

隨堂練習：統計量的計算

- 請先建造下列張量，並將之印出：
  - stat1 = torch.arange(1.0, 11.0)

- 請用下列函數，練習計算 stat1 內的各種統計量：
  - 最小元素：stat1.min()
  - 最大元素： stat1.max()
  - 總和：stat1.sum()
  - 平均值：stat1.mean()
  - 中位數： stat1.median()
  - 標準差：stat1.std()

- 參考程式碼如右圖所示：

```
tensor([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
tensor(1.)
tensor(10.)
tensor(55.)
tensor(5.5000)
tensor(5.)
tensor(3.0277)
```

```
1  import torch
2
3  stat1 = torch.arange(1.0, 11.0)
4  print(stat1)
5
6  print(stat1.min())
7  print(stat1.max())
8  print(stat1.sum())
9  print(stat1.mean())
10 print(stat1.median())
11 print(stat1.std())
```

本章總結

- **PyTorch 簡介**
  - 安裝：pip install torch
  - GPU：torch.cuda.is_available(), <Tensor>.to(<device>)
- **張量簡介**
  - 定義：PyTorch 中，儲存數據的資料結構。
  - 常見的張量資料型態：torch.int32, torch.float64, ...。
- **張量建立**
  - 一般：torch.Tensor(), torch.tensor()
  - 特殊：torch.zeros(), torch.eye(), torch.ones(), torch.full()
  - 轉換：torch.tolist(), torch.from_numpy(), <Tensor>.numpy()
- **張量資訊**
  - 軸數：.dim()
  - 維度：.size()
  - 元素：.numel()
  - 元素資料：.dtype
- **張量讀寫**
  - 底層：.storage()
  - 重塑：.view(m, n)
  - 切片：[1, 2], [:, :5], ...

本章總結

- **張量樣本點**
  - 線性：torch.arange(0.0, 5.0, 0.2)
  - 等差：torch.linspace(0.0, 5.0, 5)
  - 等比：torch.logspace(0.0, 5.0, 5)
  - 亂數：torch.randint(), torch.rand(), torch.randn(), torch.normal()
- **張量運算**
  - 負數：.neg()
  - 加減：.add(), .sub()
  - 乘法：.mul(), .matmul(), dot(), inner(), .cross(), .outer()
  - 除法：.reciprocal(), .div(), .remainder(), .inverse()
  - 壓縮擴張：.squeeze(), .unsqueeze()
  - 合併分割：.cat(), .unbind()
  - 轉置：.T
  - 小數捨入：.round(), .floor(), .ceil()
  - 指數：.pow(), .exp2(), .exp(), .sqrt()
  - 對數：.log10(), .log2(), .log()
  - 三角函數：.deg2rad(), .rad2deg(), .sin(), .cos(), .tan()
  - 統計量：.min(), .max(), .sum(), .mean(), .median(), .std()