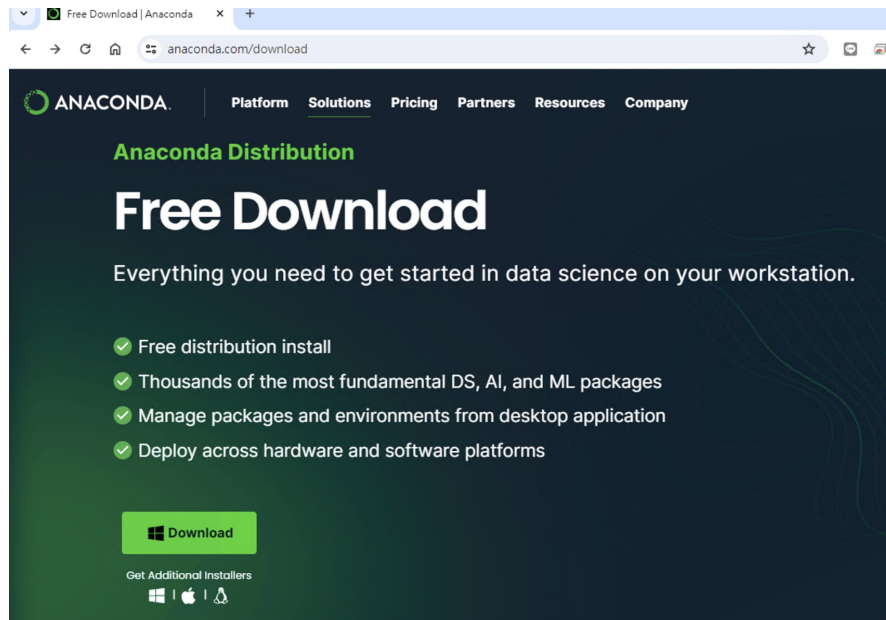


Pytorch Tensor

Anaconda

<https://www.anaconda.com/download>



`conda install pytorch torchvision`

```
Anaconda Prompt - conda install pytorch torchvision
(base) C:\Users\Ching>conda install pytorch torchvision
Collecting package metadata (current_repodata.json): done
Solving environment: / _
```

```

■ 選取 Anaconda Prompt - conda install pytorch torchvision
$ conda update -n base -c defaults conda

Or to minimize the number of packages updated during conda update use

conda install conda=24.1.2

## Package Plan ##

environment location: C:\ProgramData\anaconda3

added / updated specs:
- pytorch
- torchvision

The following packages will be downloaded:

package | build | size
-----|-----|-----
ca-certificates-2023.12.12 | haa95532_0 | 127 KB
certifi-2024.2.2 | py311haa95532_0 | 162 KB
libuv-1.44.2 | h2bbff1b_0 | 288 KB
ninja-1.10.2 | haa95532_5 | 14 KB
ninja-base-1.10.2 | h6d14046_5 | 255 KB
openssl-3.0.13 | h2bbff1b_0 | 7.4 MB
pytorch-2.2.0 | cpu_py311hd080823_0 | 112.1 MB
torchvision-0.15.2 | cpu_py311haf6e6b9_0 | 10.0 MB
typing-extensions-4.9.0 | py311haa95532_1 | 10 KB
typing_extensions-4.9.0 | py311haa95532_1 | 70 KB
-----|-----|-----
Total: | 130.5 MB

The following NEW packages will be INSTALLED:

libuv pkgs/main/win-64::libuv-1.44.2-h2bbff1b_0
ninja pkgs/main/win-64::ninja-1.10.2-haa95532_5
ninja-base pkgs/main/win-64::ninja-base-1.10.2-h6d14046_5
pytorch pkgs/main/win-64::pytorch-2.2.0-cpu_py311hd080823_0
torchvision pkgs/main/win-64::torchvision-0.15.2-cpu_py311haf6e6b9_0

The following packages will be UPDATED:

ca-certificates 2023.08.22-haa95532_0 --> 2023.12.1
certifi 2023.7.22-py311haa95532_0 --> 2024.2.2-
openssl 3.0.10-h2bbff1b_2 --> 3.0.13-h2
typing-extensions 4.7.1-py311haa95532_0 --> 4.9.0-py3
typing_extensions 4.7.1-py311haa95532_0 --> 4.9.0-py3

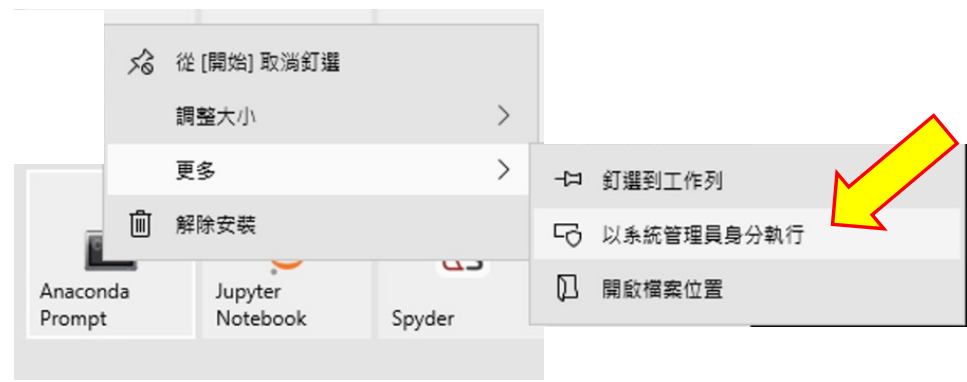
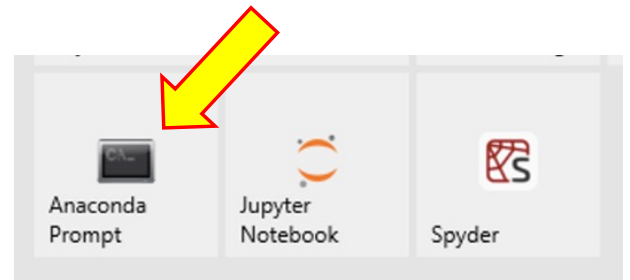
Proceed ([y]/n)? _

```

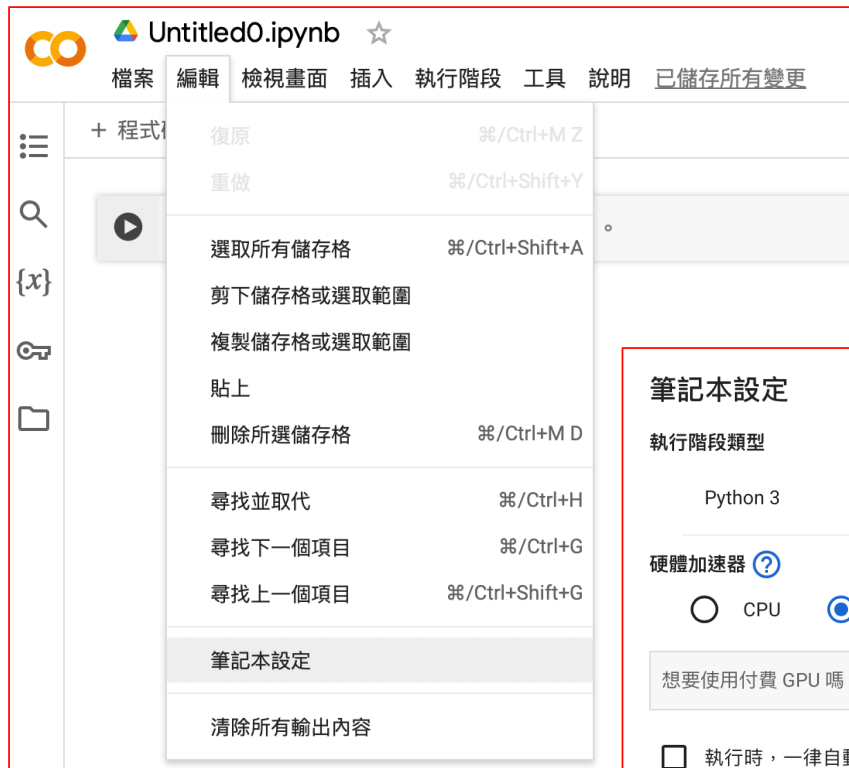
```

Proceed ([y]/n)? y
Downloading and Extracting Packages
ninja-base-1.10.2 | 255 KB | ##### 100%
typing-extensions-4. | 10 KB | ##### 100%
typing_extensions-4. | 70 KB | ##### 100%
torchvision-0.15.2 | 10.0 MB | ##### 100%
openssl-3.0.13 | 7.4 MB | ##### 100%
certifi-2024.2.2 | 162 KB | ##### 100%
ninja-1.10.2 | 14 KB | ##### 100%
pytorch-2.2.0 | 112.1 MB | ##### 100%
libuv-1.44.2 | 288 KB | ##### 100%
ca-certificates-2023 | 127 KB | ##### 100%

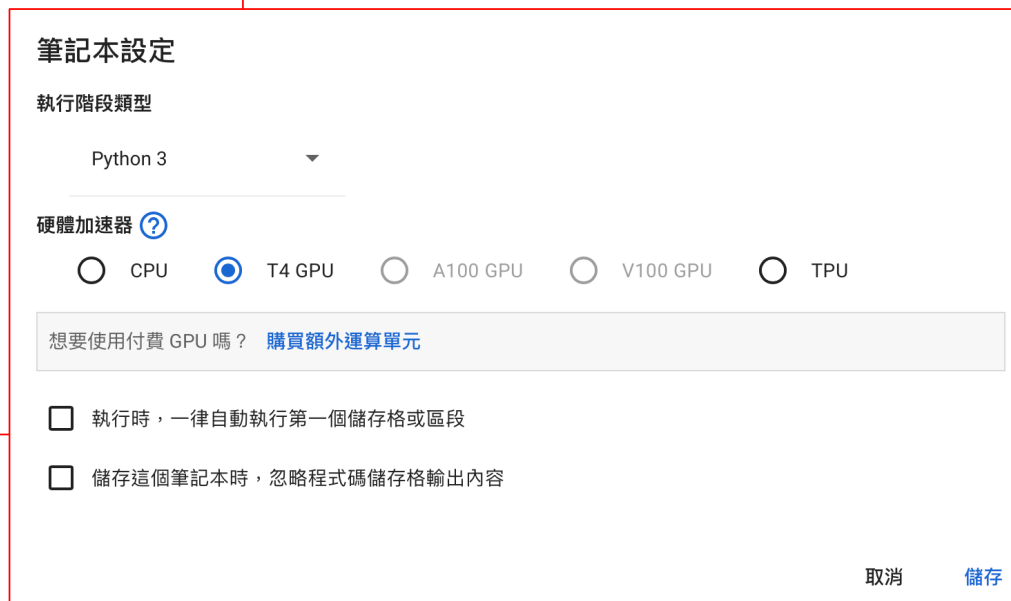
```



install PyTorch with GPU support on Google Colab



Google Colab interface showing the 'Edit' menu. The menu items include: 復原 (⌘/Ctrl+M Z), 重做 (⌘/Ctrl+Shift+Y), 選取所有儲存格 (⌘/Ctrl+Shift+A), 剪下儲存格或選取範圍, 複製儲存格或選取範圍, 貼上, 刪除所選儲存格 (⌘/Ctrl+M D), 尋找並取代 (⌘/Ctrl+H), 尋找下一個項目 (⌘/Ctrl+G), 尋找上一個項目 (⌘/Ctrl+Shift+G), 筆記本設定 (highlighted), and 清除所有輸出內容.



筆記本設定

執行階段類型

Python 3

硬體加速器 ?

CPU T4 GPU A100 GPU V100 GPU TPU

想要使用付費 GPU 嗎? [購買額外運算單元](#)

執行時，一律自動執行第一個儲存格或區段

儲存這個筆記本時，忽略程式碼儲存格輸出內容

取消 儲存

install PyTorch with GPU support on Google Colab

```
1 !pip install torch torchvision torchaudio  
2
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/site-packages (2.0.1)  
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/site-packages (0.15.2)  
Requirement already satisfied: torchaudio in /usr/local/lib/python3.10/site-packages (0.10.2)  
Requirement already satisfied: filelock in /usr/local/lib/python3.10/site-packages (3.12.2)  
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/site-packages (4.5.0)  
Requirement already satisfied: sympy in /usr/local/lib/python3.10/site-packages (1.11.1)  
Requirement already satisfied: networkx in /usr/local/lib/python3.10/site-packages (3.1)  
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/site-packages (3.1.2)  
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/site-packages (2023.1.0)  
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/site-packages (2.1.0)  
Requirement already satisfied: numpy in /usr/local/lib/python3.10/site-packages (1.25.2)  
Requirement already satisfied: requests in /usr/local/lib/python3.10/site-packages (2.31.0)  
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/site-packages (9.4.0)  
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/site-packages (2.1.2)  
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/site-packages (3.2.0)  
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/site-packages (3.4)  
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/site-packages (2.0.7)  
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/site-packages (2023.7.22)  
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/site-packages (1.3.0)
```

```
1 import torch  
2 print(torch.cuda.is_available())
```

True

```
!pip install torch torchvision torchaudio
```

```
import torch  
print(torch.cuda.is_available())
```

PyTorch Tensor

- PyTorch tensors are the fundamental data structures in PyTorch.
- They serve as powerful containers for **multi-dimensional data, enabling storage and manipulation of data of any dimensionality.**
- PyTorch tensors are extensively used in deep learning and machine learning due to their versatility and efficiency, offering features such as automatic differentiation, GPU acceleration, and convenient APIs for tensor operations.



- A scalar is a single number
- A vector is an array of numbers.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

- A matrix is a 2-D array

$$\mathbf{A} = \begin{bmatrix} A_{1,1} & A_{1,2} & \dots & A_{1,n} \\ A_{2,1} & A_{2,2} & \dots & A_{2,n} \\ \dots & \dots & \dots & \dots \\ A_{m,1} & A_{m,2} & \dots & A_{m,n} \end{bmatrix}$$

- A tensor is a n -dimensional array with $n > 2$

Source:

<https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.1-Scalars-Vectors-Matrices-and-Tensors/>

Data Type	Description	Example
torch.float32	32-bit floating point	torch.tensor([1.0])
torch.float64	64-bit floating point	torch.tensor([1.0], dtype=torch.float64)
torch.float16	16-bit floating point	torch.tensor([1.0], dtype=torch.float16)
torch.int8	8-bit integer	torch.tensor([1], dtype=torch.int8)
torch.int16	16-bit integer	torch.tensor([1], dtype=torch.int16)
torch.int32	32-bit integer	torch.tensor([1], dtype=torch.int32)
torch.int64	64-bit integer	torch.tensor([1], dtype=torch.int64)
torch.uint8	8-bit unsigned integer	torch.tensor([1], dtype=torch.uint8)
torch.bool	Boolean	torch.tensor([True], dtype=torch.bool)

Example applications of PyTorch tensors

1. Model Input and Output: In deep learning tasks, tensors are used to **represent input data** to the model, **output predictions**, and **model parameters**.

For instance, in an image classification task, an image can be represented as a tensor as input to the model, while the model's output is a tensor containing the predicted probabilities for each class.

2. Loss Function Computation: During training of deep learning models, **loss functions** are employed to quantify the discrepancy between the model predictions and the ground truth labels. Loss functions typically take tensors representing the model's output predictions and the ground truth labels as inputs.

3. Gradient Computation: PyTorch implements automatic differentiation through its Autograd mechanism, enabling automatic computation of gradients. When tensors are used in forward propagation calculations in the model, PyTorch automatically constructs a computational graph and computes gradients during backward propagation. Gradients are also represented as tensors and are used to update model parameters.

4. Data Processing and Transformation: In data preprocessing, feature engineering, and other data manipulation steps, tensors are frequently used for various operations such as normalization, standardization, dimensionality reduction, etc.

Differences between Numpy and PyTorch Tensor

1. Computation Platform:

1. **Numpy:** Numpy is a mathematical library based on Python, providing a **multidimensional array object and a set of functions for operating on these arrays**.
2. **PyTorch Tensor:** PyTorch Tensor is a tensor library designed for **deep learning**, offering functionalities similar to Numpy arrays and accelerated computation on GPUs.

2. Computation Graph and Automatic Differentiation:

1. **Numpy:** Numpy does **not support** computation graphs or automatic differentiation.
2. **PyTorch Tensor:** PyTorch Tensor supports computation graphs and automatic differentiation, making it particularly suitable for training and optimizing deep learning models.

3. GPU Acceleration:

1. **Numpy:** Numpy does not directly support GPU-accelerated computation.
2. **PyTorch Tensor:** PyTorch Tensor can utilize **GPUs for accelerated computation**, thereby speeding up model training and inference.

4. Library Ecosystem:

1. **Numpy:** Numpy has a rich ecosystem, including many libraries for scientific computing and data processing.
2. **PyTorch Tensor:** PyTorch Tensor's ecosystem is primarily focused on deep learning, providing tools and libraries for modeling and training deep neural networks.

Differences between Numpy and PyTorch Tensor

1. Creating Arrays/Tensors:

```
1 # Creating an array using Numpy
2 import numpy as np
3 numpy_array = np.array([1, 2, 3])
4 print("Numpy array:", numpy_array)
5
6 # Creating a tensor using PyTorch
7 import torch
8 torch_tensor = torch.tensor([1, 2, 3])
9 print("PyTorch tensor:", torch_tensor)
```

Differences between Numpy and PyTorch Tensor

2. Element-wise Multiplication:

```
1 # Element-wise multiplication with Numpy
2 result_np = numpy_array * 2
3 print("Numpy array multiplied by 2:", result_np)
4
5 # Element-wise multiplication with PyTorch Tensor
6 result_torch = torch_tensor * 2
7 print("PyTorch tensor multiplied by 2:", result_torch)
```

Differences between Numpy and PyTorch Tensor

- **3. Automatic Differentiation:**

```
1 # Automatic differentiation with PyTorch Tensor
2 x = torch.tensor([2.0], requires_grad=True)
3 y = x ** 2
4 y.backward()
5 print("Gradient of y w.r.t. x:", x.grad)
~
```

Tensor Creation:

- `torch.tensor()`: Creates a tensor from a Python list or array.
- `torch.zeros()`: Creates a tensor filled with zeros.
- `torch.ones()`: Creates a tensor filled with ones.
-

```
import torch
x = torch.tensor([1, 2, 3])
x
```

```
zeros_tensor = torch.zeros(2, 3)
zeros_tensor
```

```
ones_tensor = torch.ones(2, 3)
ones_tensor
```

```
▶ 1 import torch
   2 x = torch.tensor([1, 2, 3])
   3 x
```

```
↳ tensor([1, 2, 3])
```

```
[8] 1 zeros_tensor = torch.zeros(2, 3)
    2 zeros_tensor
```

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

```
[9] 1 ones_tensor = torch.ones(2, 3)
    2 ones_tensor
```

```
tensor([[1., 1., 1.],
        [1., 1., 1.]])
```

Tensor Operations:

- `torch.add()`: Adds two tensors element-wise.
- `torch.matmul()`: Performs matrix multiplication.
- `torch.sum()`: Computes the sum of tensor elements.

-

```
1 result = torch.add(x, y)
2 result = torch.matmul(matrix1, matrix2)
3 total = torch.sum(x)
```

Math Operations:

- `torch.exp()`: Computes the exponential of tensor elements.
- `torch.sin()`: Computes the sine of tensor elements.

```
import torch

# Create a tensor
x = torch.tensor([1.0, 2.0, 3.0])

# Compute the exponential of tensor
elements
exp_tensor = torch.exp(x)

print(exp_tensor)
```

```
tensor([ 2.7183,  7.3891, 20.0855])
```


Math Operations:

- `torch.mean()`: Computes the mean of tensor elements.

```
import torch

# Create a tensor
x = torch.tensor([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])

# Compute the mean of tensor elements
mean_value = torch.mean(x)
```

```
print("Tensor:")
print(x)
print("\nMean:", mean_value.item())
```

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

```
Mean: 3.5
```

Accessing values at specific positions: (1/5)

```
import torch

# Create a tensor
x = torch.tensor([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]])

# Access value at specific position
value = x[1, 2]
print("Value at position (1, 2):",
      value.item())
```

Tensor slicing: (2/5)

```
# Slice a tensor
slice_tensor = x[:, 1]
print("Sliced tensor:")
print(slice_tensor)
```

Getting/Changing tensor shape: (3/5)

```
# Get tensor shape
shape = x.shape
print("Tensor shape:", shape)
```

```
# Reshape tensor
reshaped_tensor = x.view(1, 9)
print("Reshaped tensor:")
print(reshaped_tensor)
```

Masking operations: (4/5)

```
# Masking operation
mask = x > 5
masked_tensor = x[mask]
print("Masked tensor:")
print(masked_tensor)
```

Saving/Loading tensor to file: (5/5)

```
# Save tensor to file
torch.save(x, 'tensor.pt')
print("Tensor saved successfully.")
```

```
# Load tensor from file
loaded_tensor = torch.load('tensor.pt')
print("Tensor loaded from file:")
print(loaded_tensor)
```