Anaconda Tutorial

TA : 叶俊杰

2024年9月9日星期一

Outline

- Introduction: What is Anaconda?
- Install in your computer

What is Anaconda?

Anaconda is a distribution that provides easy access to and management of packages, and unified management of environments.

Anaconda includes over 180 scientific packages and their dependencies, including conda, Python, and more.

Install Anaconda in your computer

Anaconda介绍、安装及使用教程

(https://zhuanlan.zhihu.com/p/32925500)

PyTorch Tutorial

Outline

- Background: Prerequisites & What is Pytorch?
- Training & Testing Neural Networks in Pytorch
- Dataset & Dataloader
- Tensors
- torch.nn: Models, Loss Functions
- torch.optim: Optimization
- Save/load models

Prerequisites

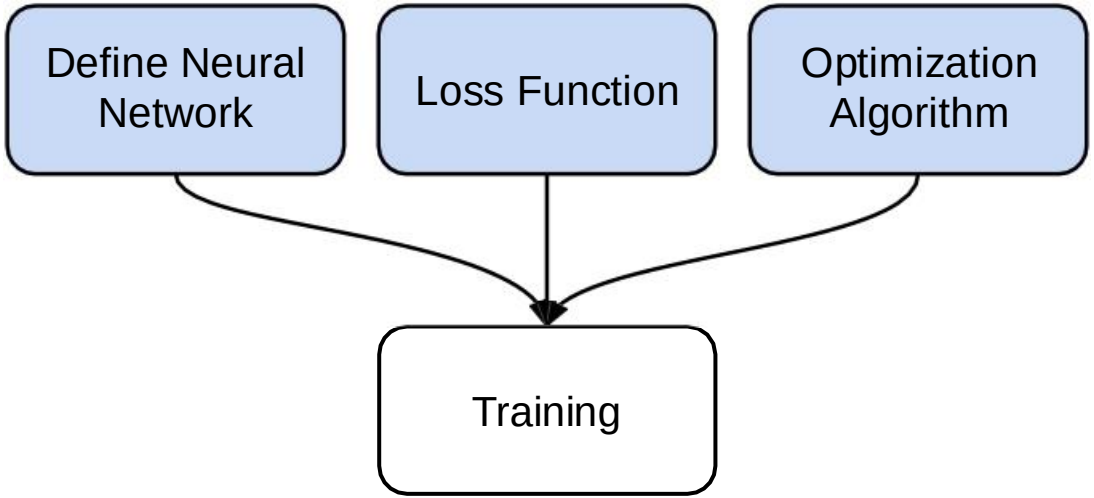
- We assume you are already familiar with…
  1. **Python3**
     - if-else, loop, function, file IO, class, …
     - refs: link1, link2, link3
  2. **Deep Learning Basics**
     - ref: link1, link2

Some knowledge of **NumPy** will also be useful!

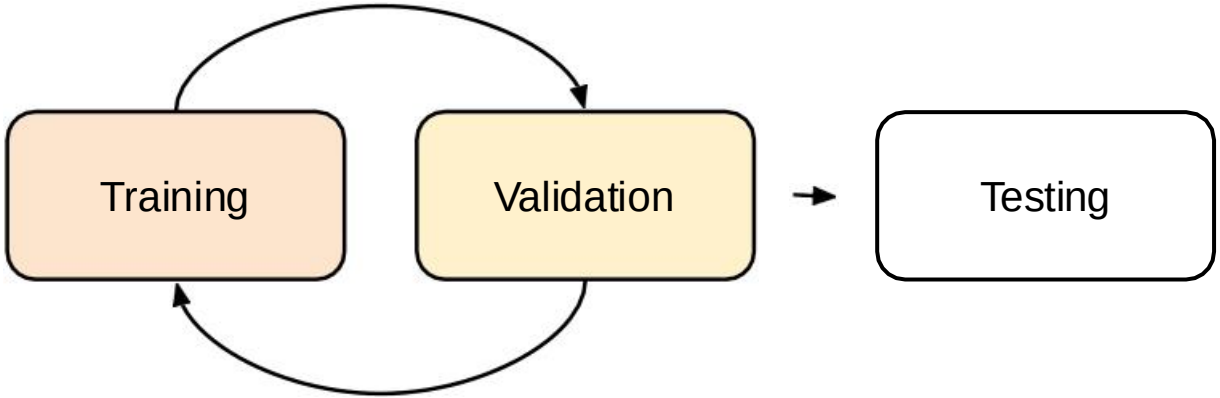
What is PyTorch?

- An **machine learning framework** in Python.
- Two main features:
  - N-dimensional **Tensor** computation (like NumPy) on **GPUs**
  - **Automatic differentiation** for training deep neural networks

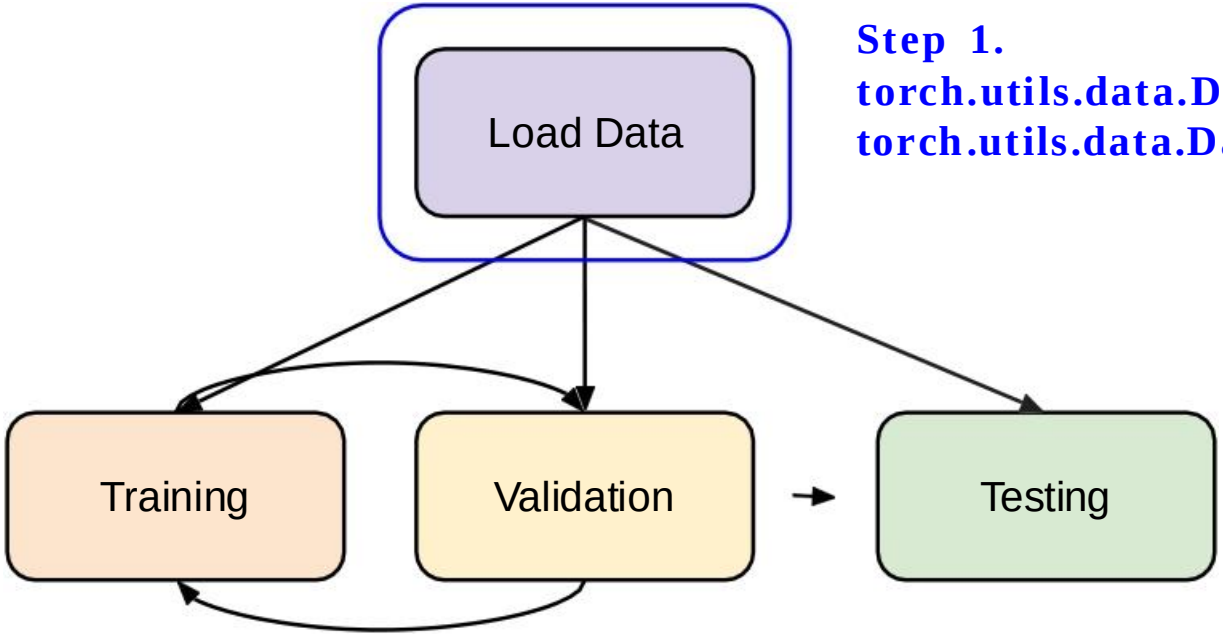
Training Neural Networks



Training & Testing Neural Networks



Training & Testing Neural Networks - in Pytorch



Step 1.
torch.utils.data.Dataset &
torch.utils.data.DataLoader

Dataset & Dataloader

- Dataset: stores data samples and expected values
- Dataloader: groups data in batches, enables multiprocessing

- **dataset** = MyDataset(file)
- dataloader = **DataLoader**(**dataset**, batch_size, **shuffle**=True)

Training: True
Testing: False

Dataset & Dataloader

```
from torch.utils.data import Dataset, DataLoader

class MyDataset(Dataset):
    def init (self, file):
        self.data = ...

    def __getitem__(self, index):
        return self.data[index]

    def __len__(self):
        return len(self.data)
```

Read data & preprocess

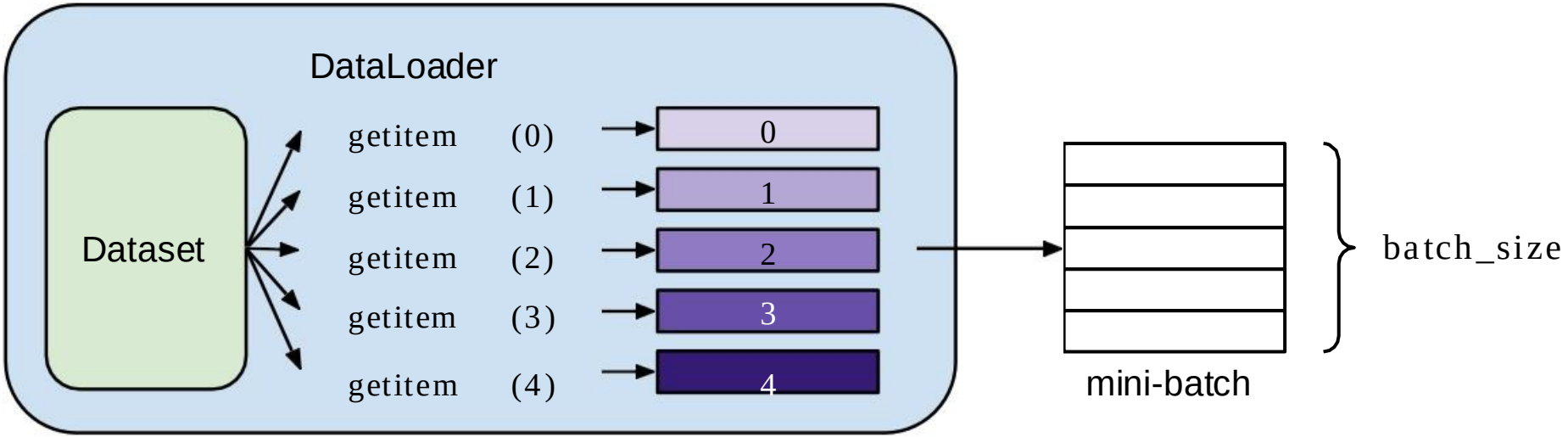
Returns one sample at a time

Returns the size of the dataset

Dataset & Dataloader

dataset = MyDataset(file)

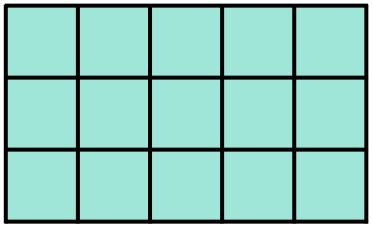
dataloader = **DataLoader**(**dataset**, batch_size=5, **shuffle**=False)



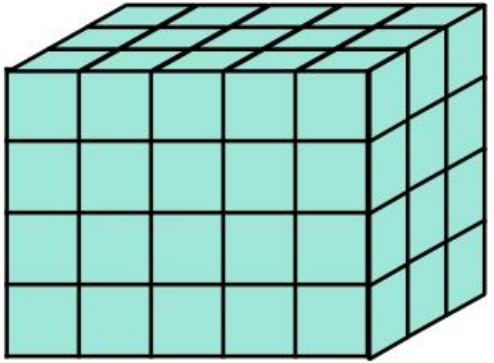
Tensors

- High-dimensional matrices (arrays)

1-D tensor
e.g. audio



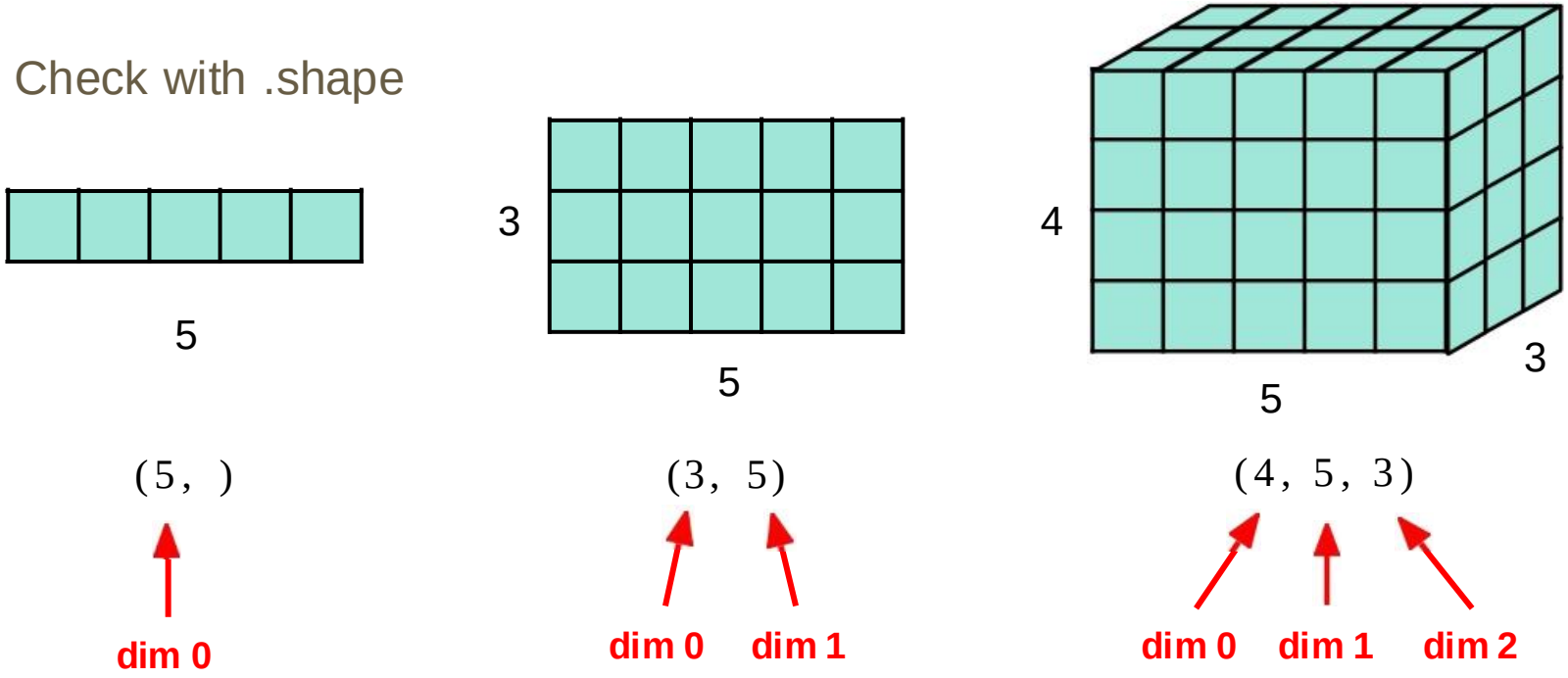
2-D tensor
e.g. black&white images



3-D tensor
e.g. RGB images

Tensors – Shape of Tensors

- Check with .shape



Note: **dim** in PyTorch == **axis** in NumPy

Tensors – Creating Tensors

- Directly from data (list or numpy.ndarray)

```
x = torch.tensor([[1, -1], [-1, 1]])
```

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

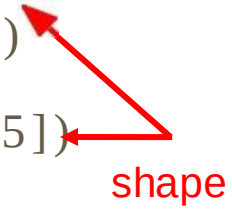
```
x = torch.from_numpy(np.array([[1, -1], [-1, 1]]))
```

- Tensor of constant zeros & ones

```
x = torch.zeros([2, 2])
```

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

```
x = torch.ones([1, 2, 5])
```




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

Tensors – Common Operations

Common arithmetic functions are supported, such as:

- Addition

```
z = x + y
```

- Subtraction

```
z = x - y
```

- Power

```
y = x.pow(2)
```

- Summation

```
y = x.sum()
```

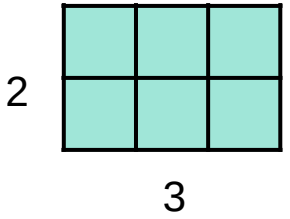
- Mean

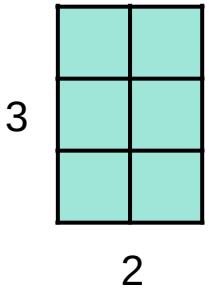
```
y = x.mean()
```

Tensors – Common Operations

- **Transpose**: transpose two specified dimensions

```
>>> x = torch.zeros([2, 3])

>>> x.shape
torch.Size([2, 3])
>>> x = x.transpose(0, 1)
>>> x.shape
torch.Size([3, 2])
```

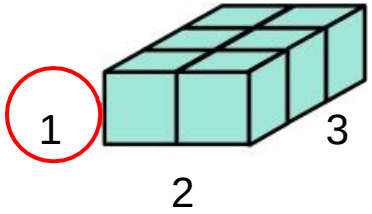


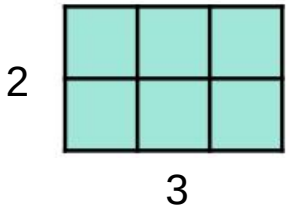


Tensors – Common Operations

- **Squeeze**: remove the specified dimension with length = 1

```
>>> x = torch.zeros([1, 2, 3])

>>> x.shape

torch.Size([1, 2, 3])

>>> x = x.squeeze(0)
                    (dim = 0)

>>> x.shape

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



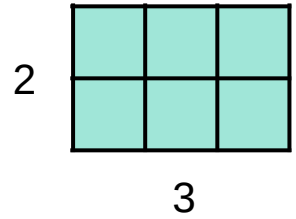


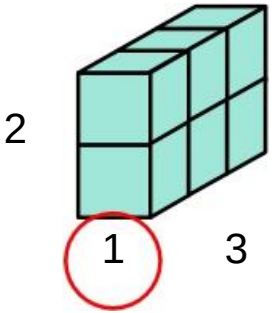
Tensors – Common Operations

- **Unsqueeze**: expand a new dimension

```
>>> x = torch.zeros([2, 3])
>>> x.shape
torch.Size([2, 3])
>>> x = x.unsqueeze(1)
>>> x.shape
torch.Size([2, 1, 3])
```

(dim = 1)

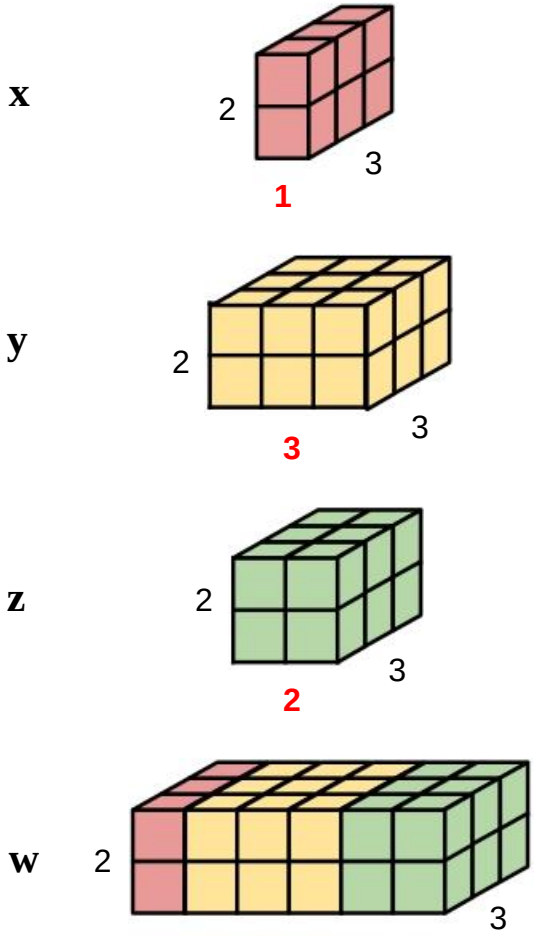




Tensors – Common Operations

- **Cat**: concatenate multiple tensors

```
>>> x = torch.zeros([2, 1, 3])
>>> y = torch.zeros([2, 3, 3])
>>> z = torch.zeros([2, 2, 3])
>>> w = torch.cat([x, y, z], dim=1)
>>> w.shape
torch.Size([2, 6, 3])
```



more operators: https://pytorch.org/docs/stable/tensors.html

Tensors – Data Type

- Using different data types for model and data will cause errors.

Data type	dtype	tensor
32-bit floating point	torch.float	torch.FloatTensor
64-bit integer (signed)	torch.long	torch.LongTensor

see official documentation for more information on data types.

Tensors – PyTorch v.s. NumPy

- Similar attributes

PyTorch	NumPy
x.shape	x.shape
x.dtype	x.dtype

see official documentation for more information on data types.

ref: https://github.com/wkentaro/pytorch-for-numpy-users

Tensors – PyTorch v.s. NumPy

- Many functions have the same names as well

PyTorch	NumPy
x.reshape / x.view	x.reshape
x.squeeze()	x.squeeze()
x.unsqueeze(1)	np.expand_dims(x, 1)

ref: https://github.com/wkentaro/pytorch-for-numpy-users

Tensors – Device

- Tensors & modules will be computed with **CPU** by default

Use **.to()** to move tensors to appropriate devices.

- CPU

```
x = x.to('cpu')
```

- GPU

```
x = x.to('cuda')
```

Tensors – Device (GPU)

- Check if your computer has NVIDIA GPU

```
torch.cuda.is_available()
```

- Multiple GPUs: specify ‘cuda:0’, ‘cuda:1’, ‘cuda:2’, ...

- Why use GPUs?
  - Parallel computing with more cores for arithmetic calculations
  - See What is a GPU and do you need one in deep learning?

Tensors – Gradient Calculation

1 `>>> x = torch.tensor([[1., 0.], [-1., 1.]],requires_grad=True)`

2 `>>> z = x.pow(2).sum()`

3 `>>> z.backward()`

4 `>>> x.grad`

```
tensor([[ 2.,   0.],
        [-2.,   2.]])
```

1 $$x = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$$

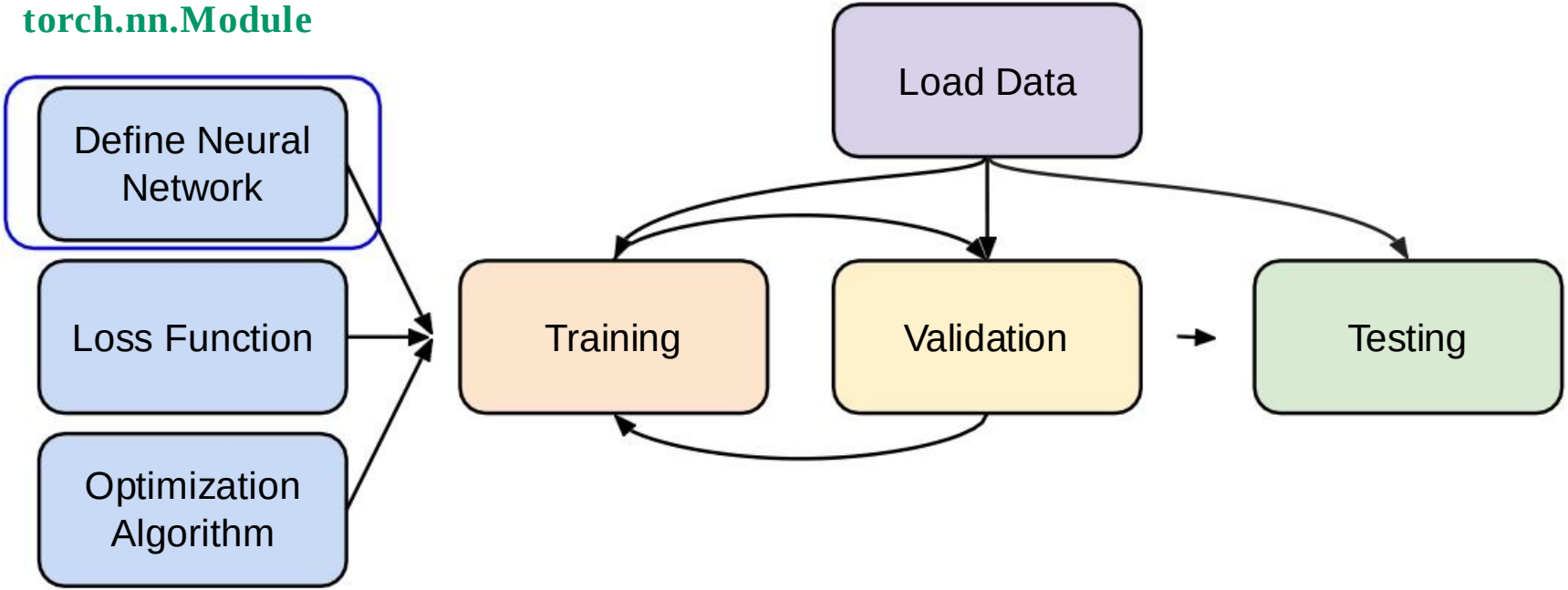
2 $$z = \sum_i \sum_j x_{i,j}^2$$

3 $$\frac{\partial z}{\partial x_{i,j}} = 2x_{i,j}$$

4 $$\frac{\partial z}{\partial x} = \begin{bmatrix} 2 & 0 \\ -2 & 2 \end{bmatrix}$$

Training & Testing Neural Networks – in Pytorch

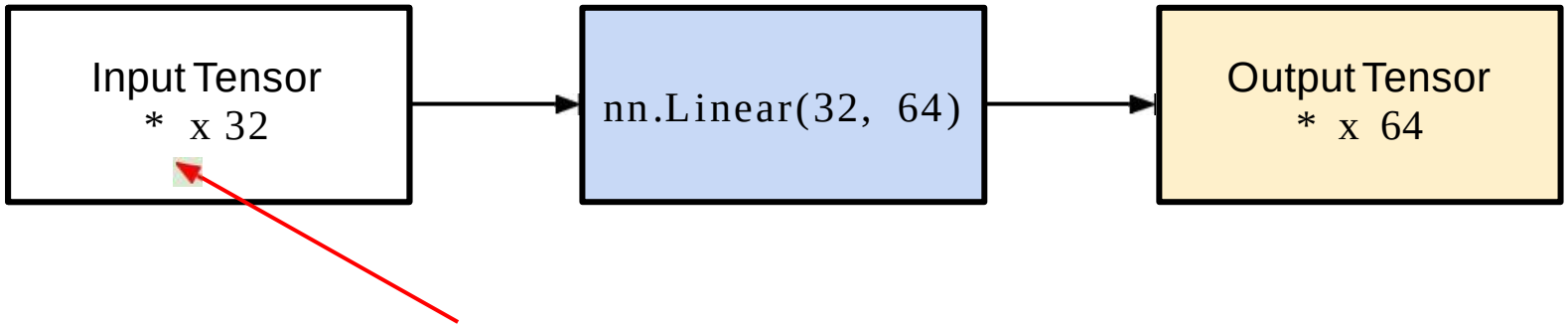
Step 2.
torch.nn.Module



torch.nn – Network Layers

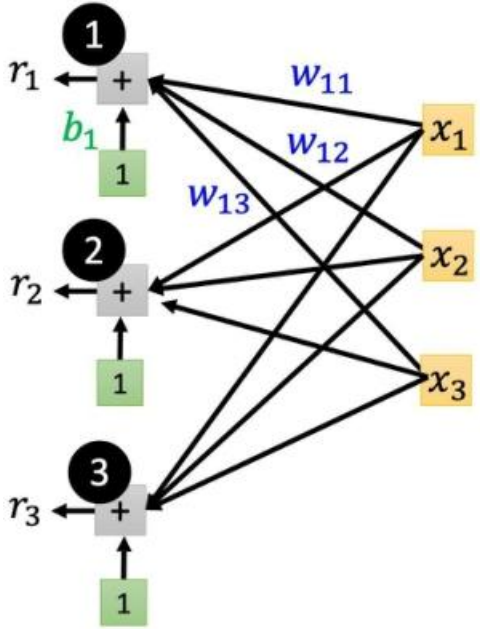
- Linear Layer (**Fully-connected** Layer)

```
nn.Linear(in_features, out_features)
```



torch.nn – Network Layers

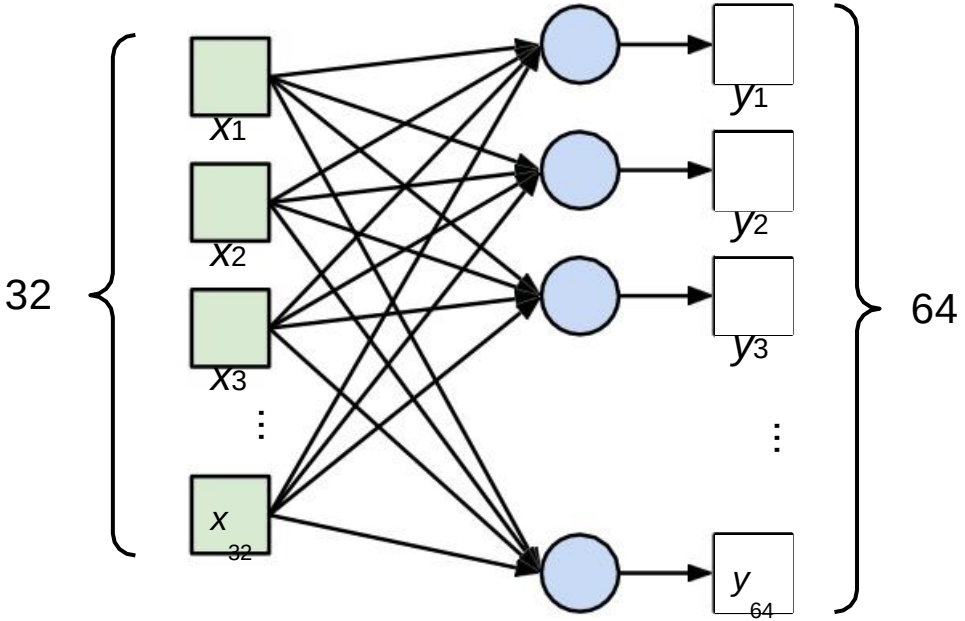
- Linear Layer (**Fully-connected** Layer)



$$\boldsymbol{b} + \boldsymbol{W}\,\boldsymbol{x}$$

torch.nn – Neural Network Layers

- Linear Layer (**Fully-connected** Layer)

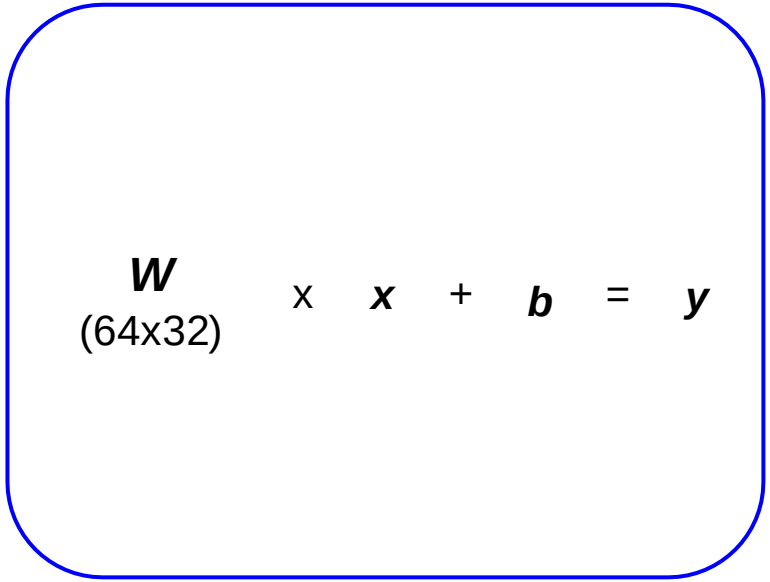


$$\underset{(64\times32)}{\boldsymbol{W}} \times \boldsymbol{x} + \boldsymbol{b} = \boldsymbol{y}$$

torch.nn – Network Parameters

- Linear Layer (**Fully-connected** Layer)

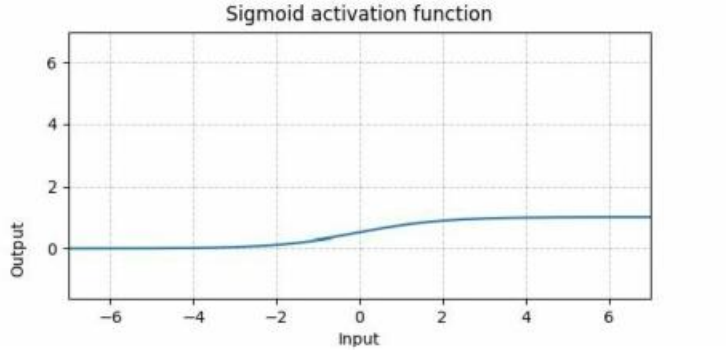
```
>>> layer = torch.nn.Linear(32, 64)
>>>  layer.weight.shape
torch.Size([64, 32])
>>>  layer.bias.shape
torch.Size([64])
```



torch.nn – Non-Linear Activation Functions

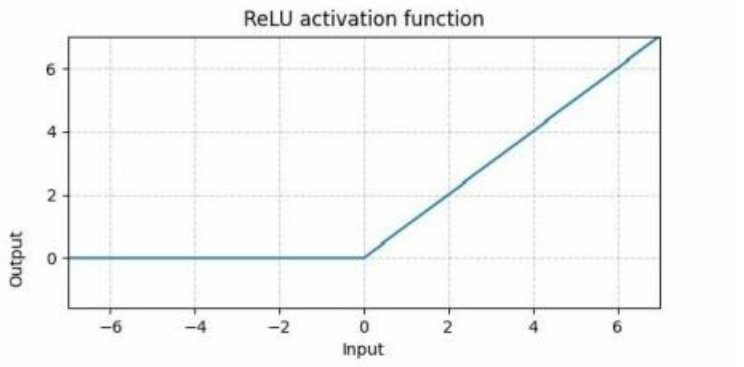
- Sigmoid Activation

  `nn.Sigmoid()`



- ReLU Activation

  `nn.ReLU()`



torch.nn – Build your own neural network

```
import torch.nn as nn

class MyModel(nn.Module):
    def init(self):
        super(MyModel, self).init ()
        self.net = nn.Sequential(
            nn.Linear(10, 32),
            nn.Sigmoid(),
            nn.Linear(32, 1)
        )

    def forward(self, x):
        return self.net(x)
```

Initialize your model & define layers

Compute output of your NN

torch.nn – Build your own neural network

```
import torch.nn as nn
```

```
class MyModel(nn.Module):
    Def __init__(self):
        super(MyModel,
        self).__init__ () self.net =
        nn.Sequential(
            nn.Linear(10, 32),
            nn.Sigmoid(),
            nn.Linear(32, 1)
        )

    def forward(self, x):
        return self.net(x)
```

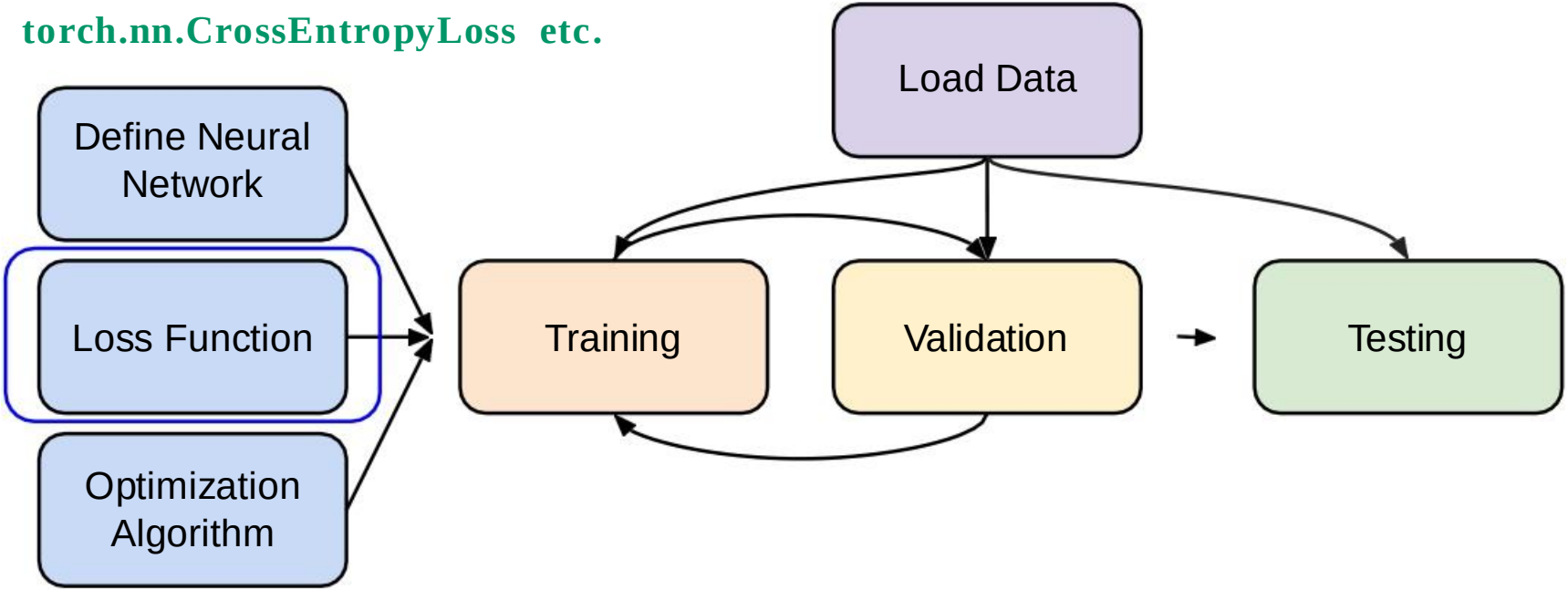
=

```
import torch.nn as nn
```

```
class MyModel(nn.Module):
    def _init_(self):
        super(MyModel, self)._ init_()
        self.layer1 = nn.Linear(10, 32)
        self.layer2 = nn.Sigmoid()
        self.layer3 = nn.Linear(32,1)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        return out
```

Training & Testing Neural Networks – in Pytorch

Step 3.
torch.nn.MSELoss
torch.nn.CrossEntropyLoss etc.



torch.nn – Loss Functions

- Mean Squared Error (for regression tasks)

```
criterion = nn.MSELoss()
```

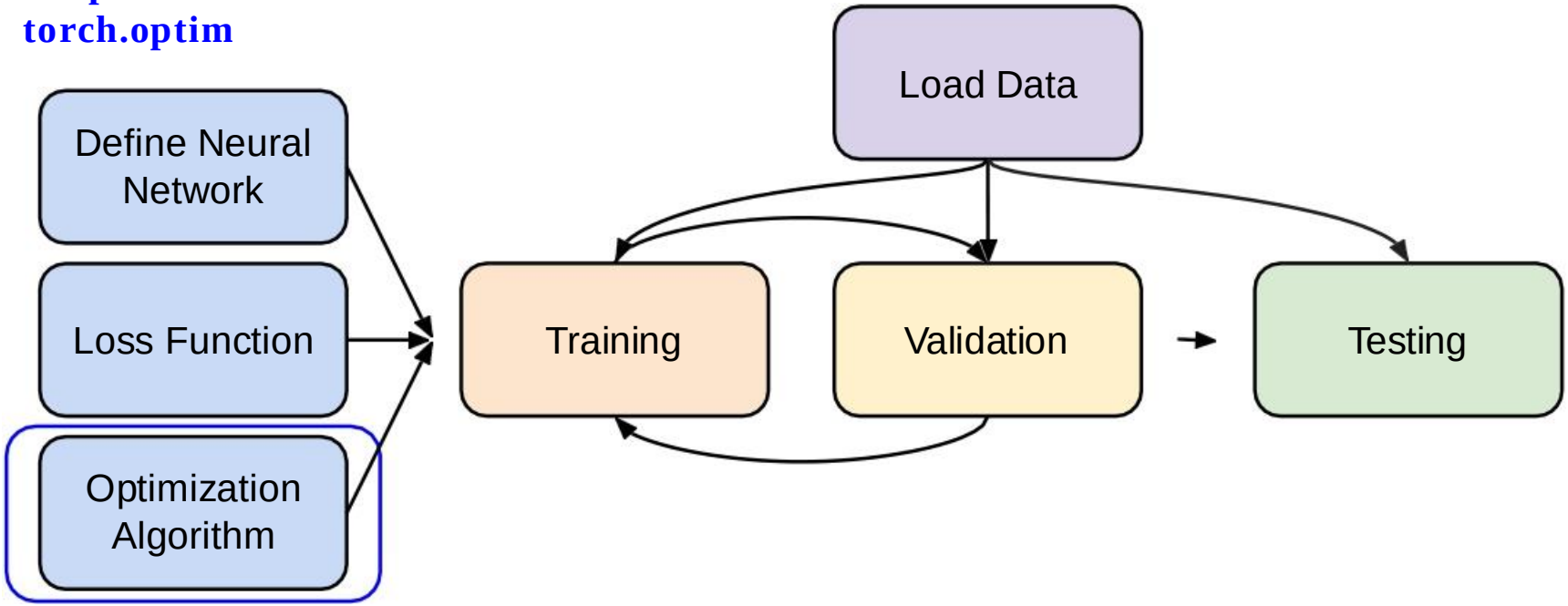
- Cross Entropy (for classification tasks)

```
criterion = nn.CrossEntropyLoss()
```

- `loss = criterion(model_output, expected_value)`

Training & Testing Neural Networks – in Pytorch

Step 4.
torch.optim



torch.optim

- Gradient-based **optimization algorithms** that adjust network parameters to reduce error. (See Adaptive Learning Rate lecture video)

- E.g. Stochastic Gradient Descent (SGD)

```
torch.optim.SGD(model.parameters(), lr, momentum = 0)
```

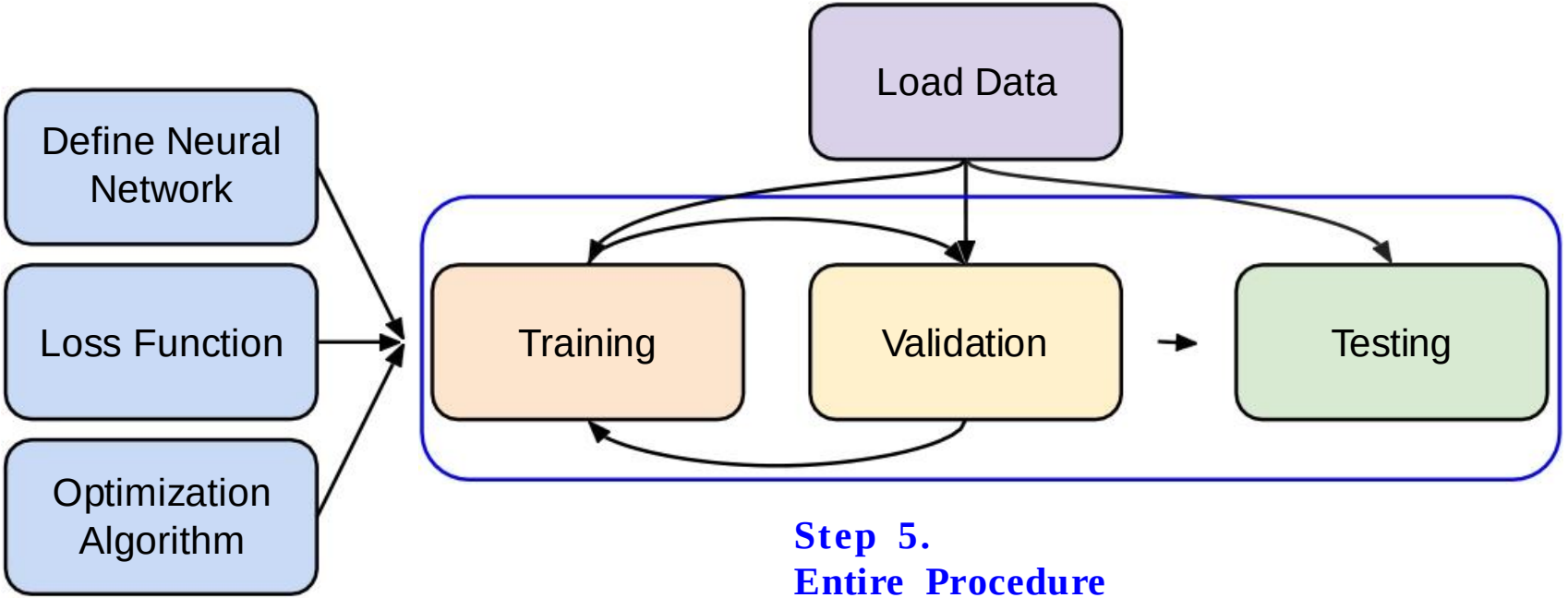
torch.optim

```
optimizer = torch.optim.SGD(model.parameters(), lr, momentum = 0)
```

- For every batch of data:
  1. Call `optimizer.zero_grad()` to reset gradients of model parameters.
  2. Call `loss.backward()` to backpropagate gradients of prediction loss.
  3. Call `optimizer.step()` to adjust model parameters.

See official documentation for more optimization algorithms.

Training & Testing Neural Networks – in Pytorch



Step 5.
Entire Procedure

Neural Network Training Setup

```
dataset = MyDataset(file)
```
read data via MyDataset

```
tr_set = DataLoader(dataset, 16, shuffle=True)
```
put dataset into Dataloader

```
model = MyModel().to(device)
```
construct model and move to device (cpu/cuda)

```
criterion = nn.MSELoss()
```
set loss function

```
optimizer = torch.optim.SGD(model.parameters(), 0.1)
```
set optimizer

Neural Network Training Loop

```
for epoch in range(n_epochs):
    model.train()
    for x, y in tr_set:
        optimizer.zero_grad()
        x, y = x.to(device), y.to(device)
        pred = model(x)
        loss = criterion(pred, y)
        loss.backward() optimizer.step()
```

iterate n_epochs

set model to train mode

iterate through the dataloader

set gradient to zero

move data to device (cpu/cuda)

forward pass (compute output)

compute loss

compute gradient (backpropagation)

update model with optimizer

Neural Network Validation Loop

```
model.eval()                                        set model to evaluation mode
total_loss = 0
for x, y in dv_set:                                 iterate through the dataloader
    x, y = x.to(device), y.to(device)               move data to device (cpu/cuda)
    with torch.no_grad():                           disable gradient calculation
        pred = model(x)                             forward pass (compute output)
        loss = criterion(pred, y)                   compute loss
    total_loss += loss.cpu().item() * len(x)        accumulate loss
avg_loss = total_loss / len(dv_set.dataset)         compute averaged loss
```

Neural Network Testing Loop

```
model.eval()
preds = []
for x in tt_set:
    x = x.to(device)
    with torch.no_grad():
        pred = model(x)
        preds.append(pred.cpu())
```

set model to evaluation mode

iterate through the dataloader

move data to device (cpu/cuda)

disable gradient calculation

forward pass (compute output)

collect prediction

Notice - model.eval(), torch.no_grad()

- **model.eval()**
  Changes behaviour of some model layers, such as dropout and batch normalization.

- **with torch.no_grad()**
  Prevents calculations from being added into gradient computation graph.
  Usually used to prevent accidental training on validation/testing data.

Save/Load Trained Models

- Save

```
torch.save(model.state_dict(), path)
```

- Load

```
ckpt = torch.load(path)
model.load_state_dict(ckpt)
```

More About PyTorch

- torchaudio
  - speech/audio processing
- torchtext
  - natural language processing
- torchvision
  - computer vision
- skorch
  - scikit-learn + pyTorch

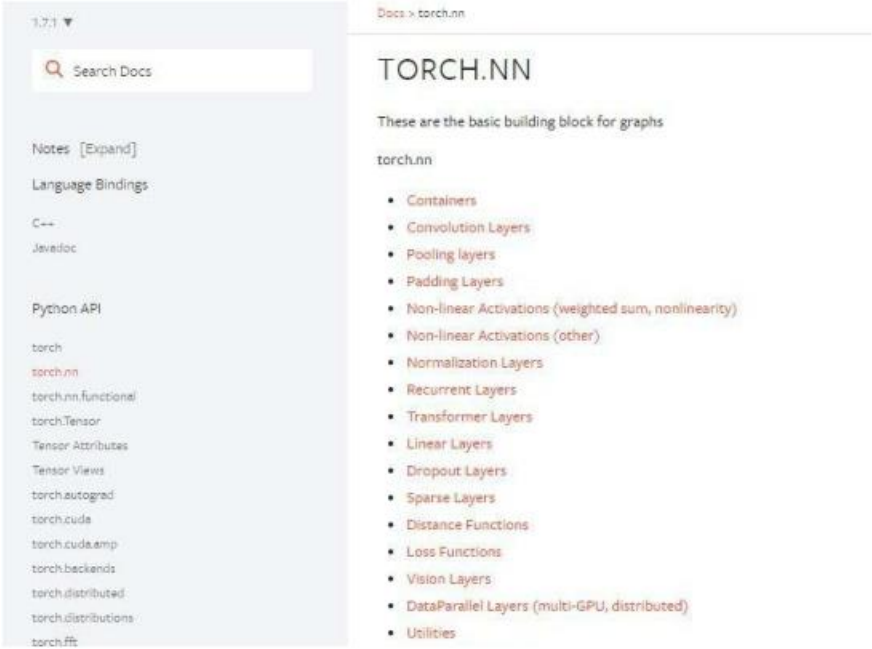
More About PyTorch

- Useful github repositories using PyTorch
  - Huggingface Transformers (transformer models: BERT, GPT, ...)
  - Fairseq (sequence modeling for NLP & speech)
  - ESPnet (speech recognition, translation, synthesis, ...)
  - Most implementations of recent deep learning papers
  - …

PyTorch Documentation

https://pytorch.org/docs/stable/

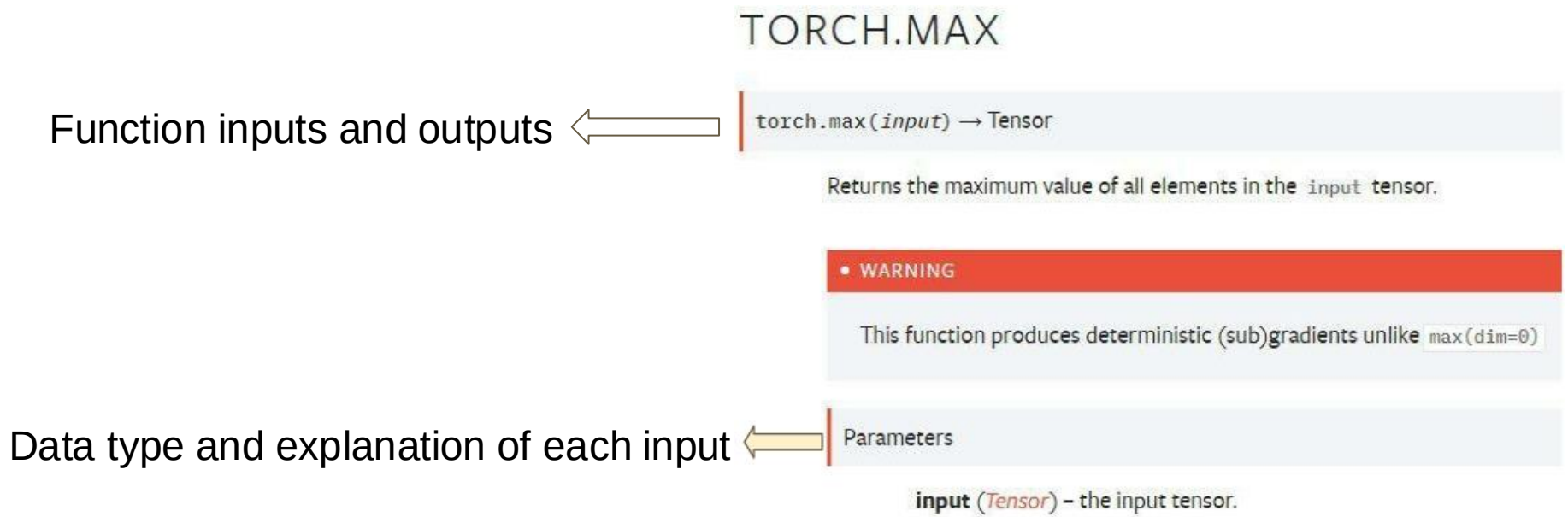
- torch.nn -> Neural Network
- torch.optim -> Optimization Algorithms
- torch.utils.data -> Dataset, Dataloader



PyTorch Documentation Example

Function inputs and outputs

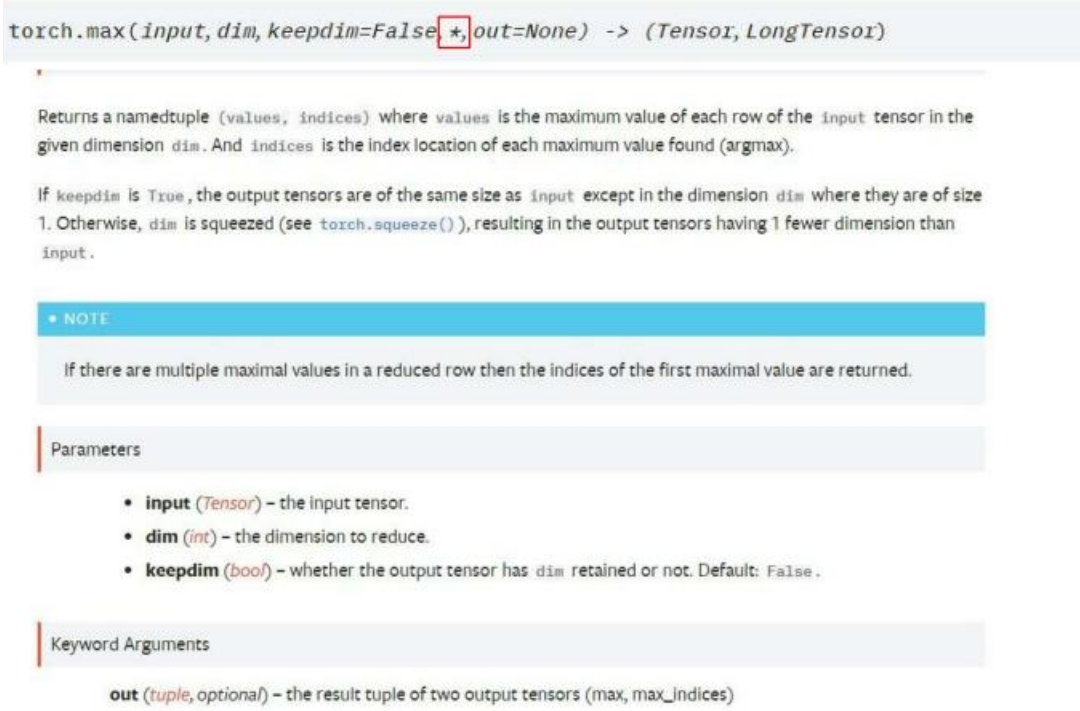
Data type and explanation of each input



PyTorch Documentation Example

- Some functions behave differently with different inputs
- Parameters : You don't need to specify the name of the argument (Positional Arguments)
- Keyword Arguments : You have to specify the name of the argument

*They are separated by **



`torch.max(input, dim, keepdim=False, *, out=None) -> (Tensor, LongTensor)`

Returns a namedtuple `(values, indices)` where `values` is the maximum value of each row of the `input` tensor in the given dimension `dim`. And `indices` is the index location of each maximum value found (argmax).

If `keepdim` is `True`, the output tensors are of the same size as `input` except in the dimension `dim` where they are of size 1. Otherwise, `dim` is squeezed (see `torch.squeeze()`), resulting in the output tensors having 1 fewer dimension than `input`.

NOTE

If there are multiple maximal values in a reduced row then the indices of the first maximal value are returned.

Parameters

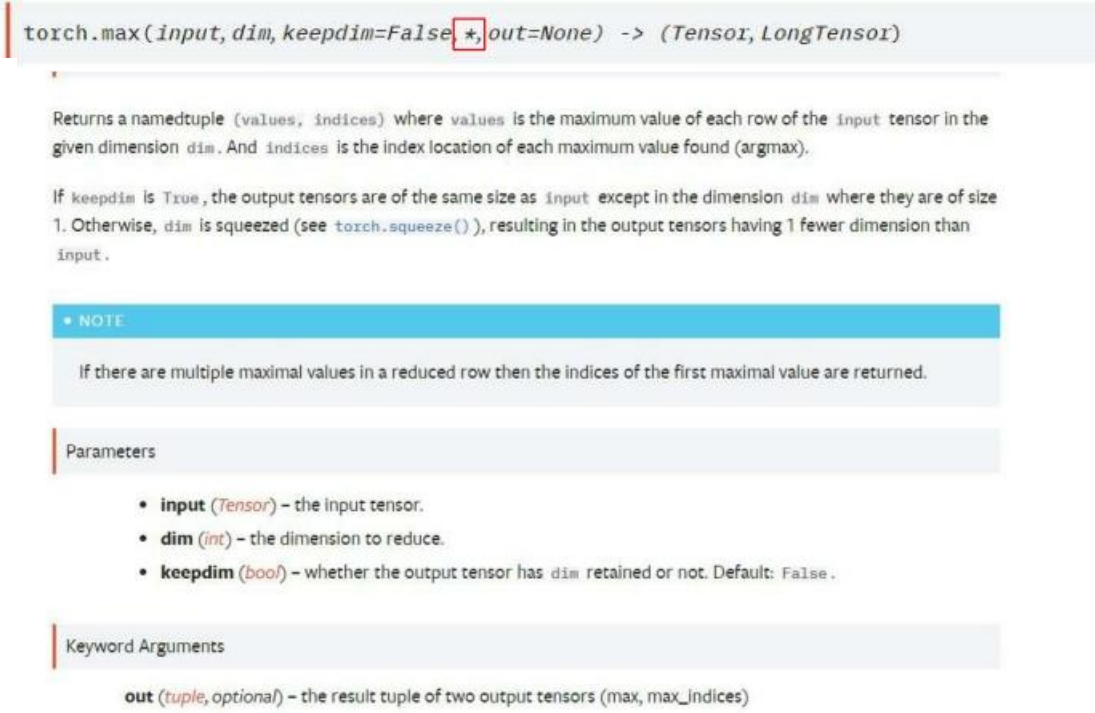
- **input** (*Tensor*) – the input tensor.
- **dim** (*int*) – the dimension to reduce.
- **keepdim** (*bool*) – whether the output tensor has `dim` retained or not. Default: `False`.

Keyword Arguments

out (*tuple, optional*) – the result tuple of two output tensors (max, max_indices)

PyTorch Documentation Example

- Some functions behave differently with different inputs
- Arguments with default value : Some arguments have a default value (keepdim=False), so passing a value of this argument is optional



`torch.max(input, dim, keepdim=False, *, out=None) -> (Tensor, LongTensor)`

Returns a namedtuple `(values, indices)` where `values` is the maximum value of each row of the `input` tensor in the given dimension `dim`. And `indices` is the index location of each maximum value found (argmax).

If `keepdim` is `True`, the output tensors are of the same size as `input` except in the dimension `dim` where they are of size 1. Otherwise, `dim` is squeezed (see `torch.squeeze()`), resulting in the output tensors having 1 fewer dimension than `input`.

• NOTE

If there are multiple maximal values in a reduced row then the indices of the first maximal value are returned.

Parameters

- **input** (*Tensor*) – the input tensor.
- **dim** (*int*) – the dimension to reduce.
- **keepdim** (*bool*) – whether the output tensor has `dim` retained or not. Default: `False`.

Keyword Arguments

out (*tuple, optional*) – the result tuple of two output tensors (max, max_indices)

PyTorch Documentation Example

Three Kinds of torch.max

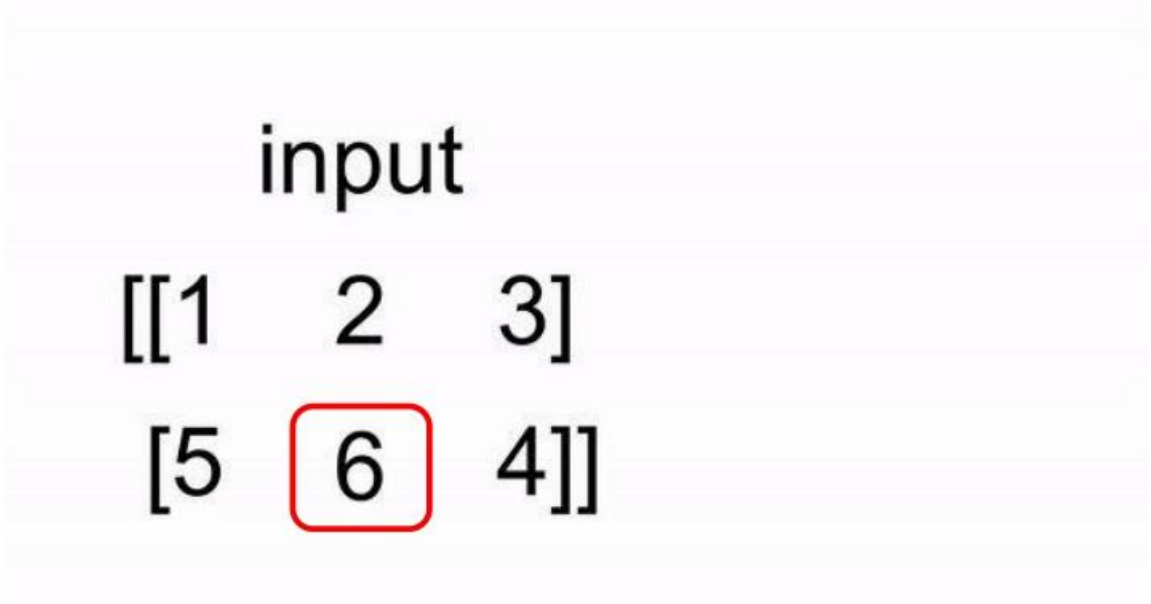
1. **`torch.max(input) → Tensor`**
2. **`torch.max(input, dim, keepdim=False, *, out=None) → (Tensor, LongTensor)`**
3. **`torch.max(input, other, *, out=None) → Tensor`**

`input : Tensor, dim : int, keepdim : bool`
`other : Tensor`

PyTorch Documentation Example

1.`torch.max(input)` → `Tensor`

Find the maximum value of a tensor, and return that value.



PyTorch Documentation Example

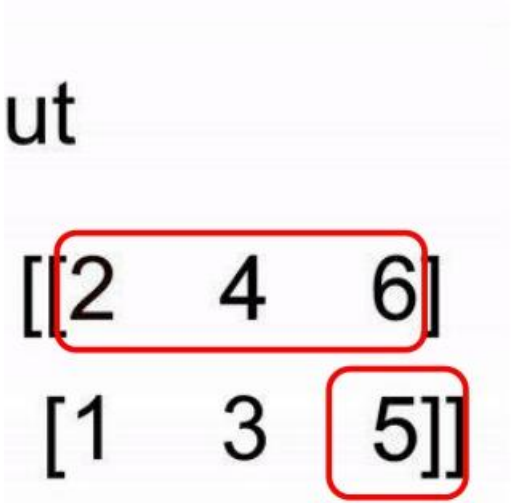
2. torch.max(input, dim, keepdim=False, *, out=None) → (Tensor, LongTensor)

Find the maximum value of a tensor along a dimension, and return that value, along with the index corresponding to that value.

PyTorch Documentation Example

`3.torch.max(input, other) → Tensor`

Perform element-wise comparison between two tensors of the same size, and select the maximum of the two to construct a tensor with the same size.



Common Errors - torch.max (Colab)

Three Kinds of torch.max

1. `torch.max(input)` → `Tensor`
2. `torch.max(input, dim, keepdim=False, *, out=None)` → `(Tensor, LongTensor)`
3. `torch.max(input, other, *, out=None)` → `Tensor`

```
input : Tensor
dim : int
keepdim : bool
other : Tensor
```

Colab code

```
x = torch.randn(4,5)
y = torch.randn(4,5)
m, idx = torch.max(x,0,False,p)→x
     *out is a keyword argument
m, idx = torch.max(x,True)→x
     *did not specify dim
```

Common Errors – Tensor on Different Device to Model

```
model = torch.nn.Linear(5,1).to("cuda:0")
x = torch.randn(5).to("cpu")
y = model(x)
```

Tensor for * is on CPU, but expected them to be on GPU

=> send the tensor to GPU

```
x = torch.randn(5).to("cuda:0")
y = model(x)
print(y.shape)
```

Common Errors - Mismatched Dimensions

```
x = torch.randn(4,5)
y = torch.randn(5,4)
z = x + y
```

The size of tensor a (5) must match the size of tensor b (4) at non-singleton dimension 1

=> the shape of a tensor is incorrect, use **`transpose`**, **`squeeze`**, **`unsqueeze`** to align the dimensions

```
y = y.transpose(0,1)
z = x + y
print(z.shape)
```

Common Errors - Cuda Out of Memory

```
import torch
import torchvision.models as models
resnet18 = models.resnet18().to( "cuda:0" ) # Neural Networks for Image
data = torch.randn( 512,3,244,244) # Create fake data (512
out = resnet18(data.to( "cuda:0" )) # Use Data as Input and Feed to
print(out.shape)
```

CUDA out of memory. Tried to allocate 350.00 MiB (GPU 0; 14.76 GiB total capacity; 11.94 GiB already allocated; 123.75 MiB free; 13.71 GiB reserved in total by PyTorch)

=> The batch size of data is too large to fit in the GPU. Reduce the batch size.

Common Errors - Mismatched Tensor Type

```
import torch.nn as nn
L = nn.CrossEntropyLoss()
outs = torch.randn(5,5)
labels = torch.Tensor([1,2,3,4,0])
lossval = L(outs, labels) # Calculate CrossEntropyLoss between outs and
labels
```

expected scalar type Long but found Float

=> labels must be long tensors, cast it to type “Long” to fix this issue

```
labels = labels.long()
lossval = L(outs,labels)
print(lossval)
```

Colab Tutorial

Outline

- Introduction
- Getting Started
- Changing Runtime
- Executing Code Block
- Check GPU type
- File Manipulation
- Mounting Google Drive
- Saving Notebook
- Useful Linux Commands
- Problems You May Encounter… (very important)
- References

Introduction

What is Colab?

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

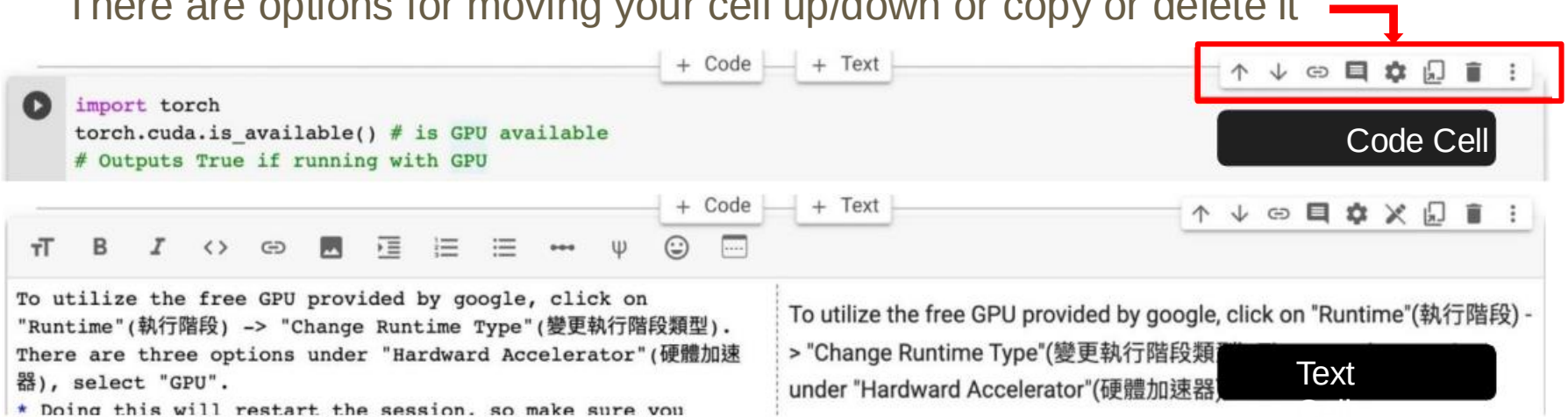
- Zero configuration required
- Free access to GPUs
- Easy sharing

Getting Started

Creating a new cell

You can create a new code cell by clicking on +Code, clicking on +Text generates a text cell

There are options for moving your cell up/down or copy or delete it



Getting Started

You can type python code in the code cell, or use a leading exclamation mark ! to change the code cell to treating the input as a shell script

```
[ ]  import torch
     torch.cuda.is_available() # is GPU available
     # Outputs True if running with GPU
```

→ **python**

```
[ ]  # List all the files under the working directory
     !ls
```

→ **shell script**

Getting Started

Using an exclamation mark (!) starts a new shell, does the operations, and then kills that shell, while percentage (%) affects the process associated with the notebook, and it is called a magic command.

Use % instead of ! for cd (change directory) command

other magic commands are listed here

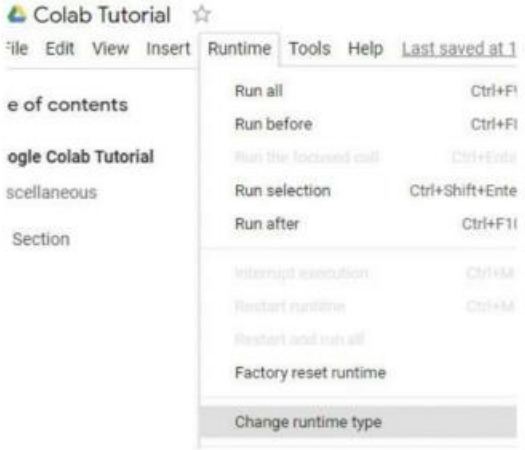
Changing Runtime

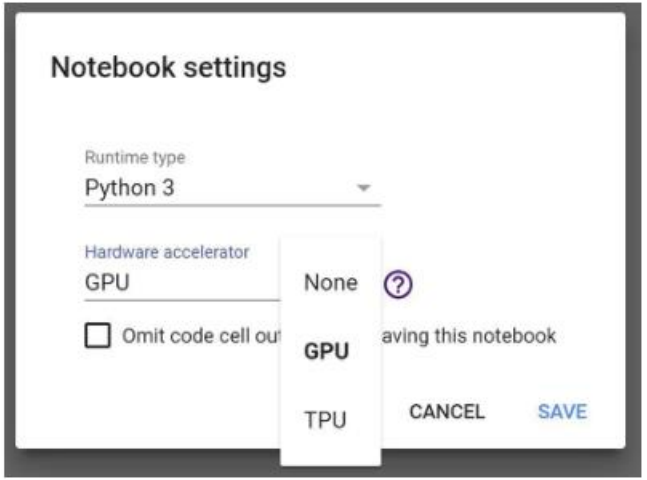
To utilize the free GPU provided by google,

click on "Runtime"(執行階段) → "Change Runtime Type"(變更執行階段類型).

select "**GPU**" for "Hardware Accelerator"(硬體加速器)

Doing this will restart the session, so make sure you change to the desired runtime before executing any code.





Executing Code Block

Click on the play button to execute a specific code cell

```
import torch
torch.cuda.is_available() # is GPU available
# Outputs True if running with GPU
```

Executing Code Block

Other options to run your code

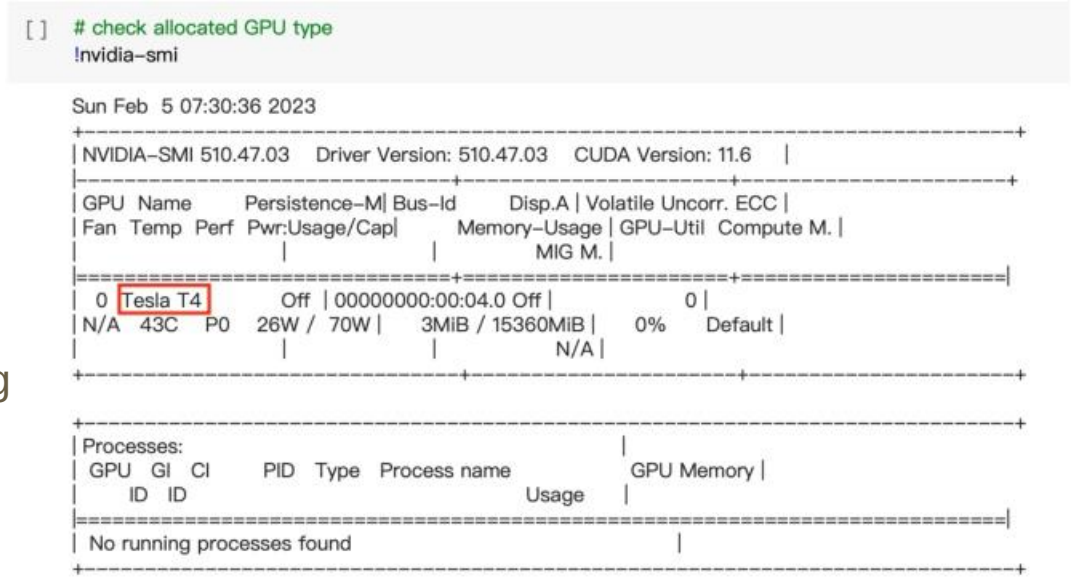
Check GPU Type

Use the command **nvidia-smi** to check the allocated GPU type

Available GPUs:

T4 > K80

(but most of the time you get K80 using the free Colab)



File Manipulation

Download files via Google Drive

1. Download Files via google drive

A file stored in Google Drive has the following sharing **link**：

https://drive.google.com/file/d/14FK5G6DOh7EdLyoj4D5teRSzriTOUPD7/view?usp=sharing

It is possible to download the file via Colab knowing the **link**, using the **--fuzzy** command.

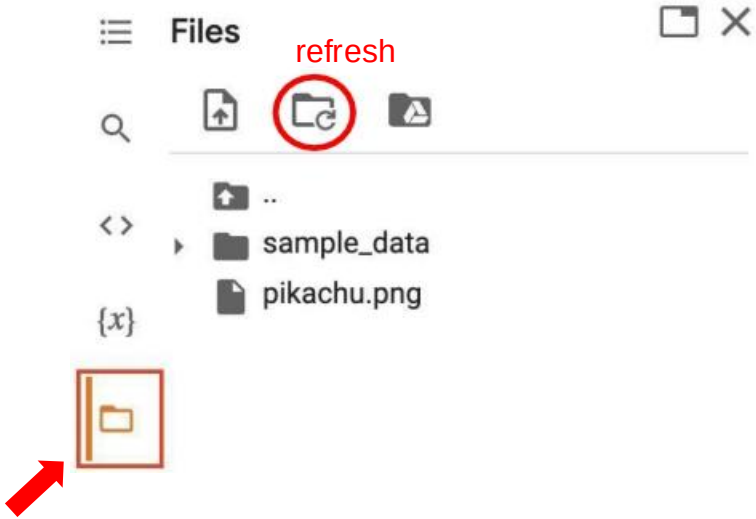
```
[ ]  # Download the file with the following link,  and rename it to pikachu.png
     !gdown --fuzzy https://drive.google.com/file/d/14FK5G6DOh7EdLyoj4D5teRSzriTOUPD7/view?usp=sharing --output pikachu.png
```

```
Downloading...
From: https://drive.google.com/uc?id=14FK5G6DOh7EdLyoj4D5teRSzriTOUPD7
To: /content/pikachu.png
100% 890k/890k [00:00<00:00, 155MB/s]
```

File Manipulation

File Structure

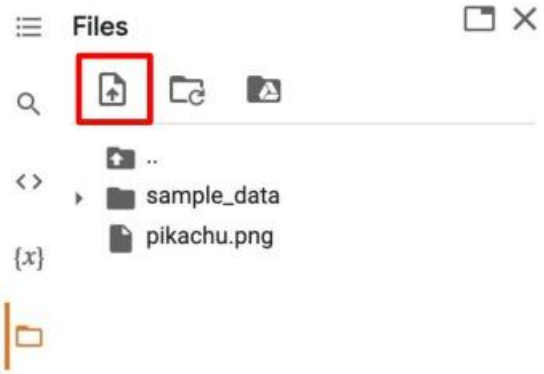
- You may click on the folder icon on the left to view your current files
- After downloading files, if the files are not immediately shown, click the refresh button
- Files are temporarily stored, and will be removed once you end your session.



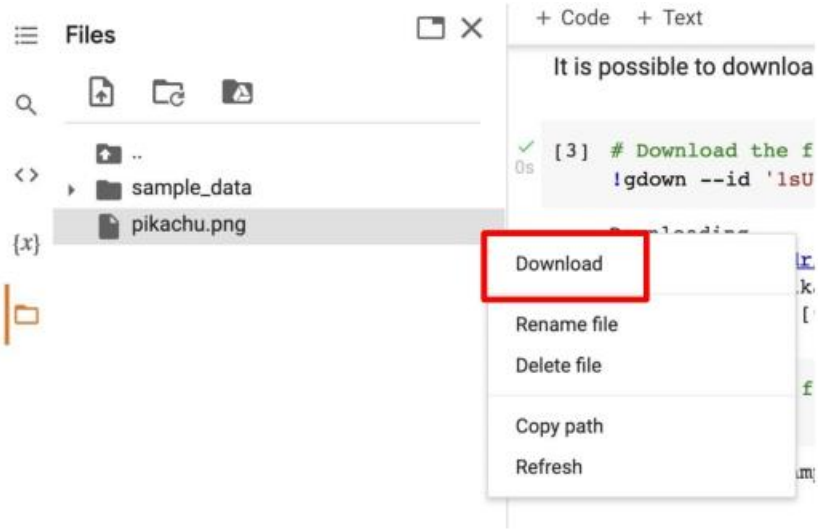
File Manipulation

Upload and Download Files

Click the upload icon to upload local files to your session



click ⋮ to download files to your local



Mounting Google Drive

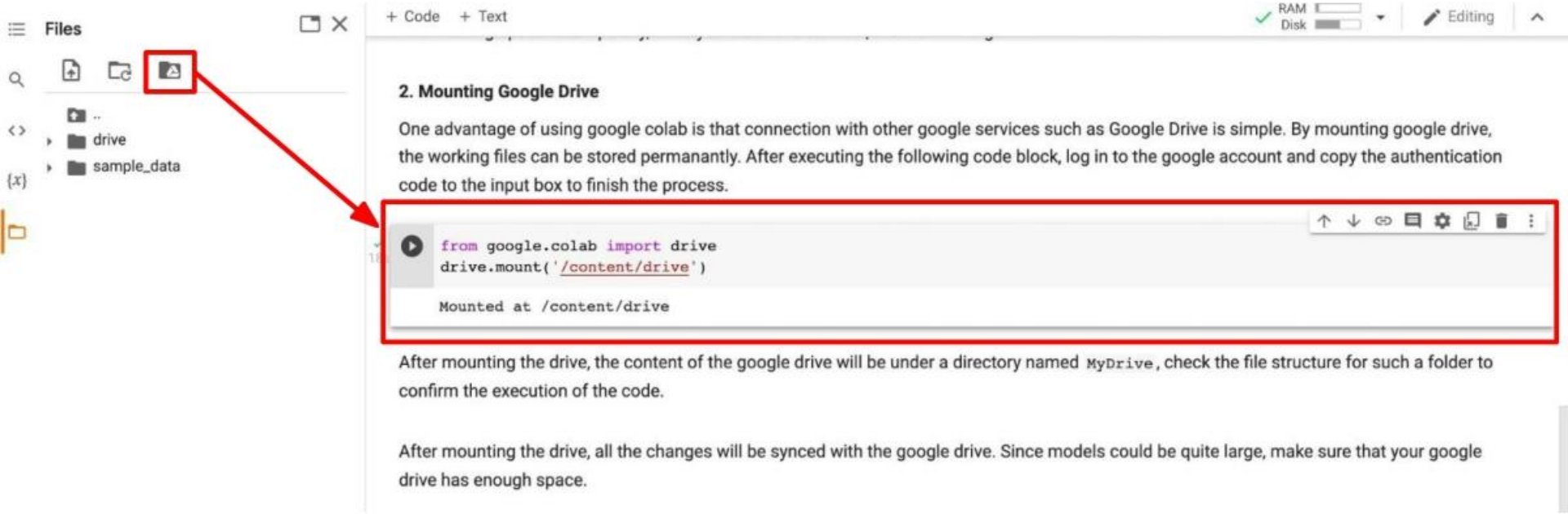
If you don’t want to download the data every time you start a new session, or you want some files to be saved permantly,

you can mount your own google drive to colab and directly download/save the data to your google drive.

Mount Drive:

Mounting Google Drive

Click on the Google Drive icon, the **Mount Drive** code block will be generated



Mounting Google Drive

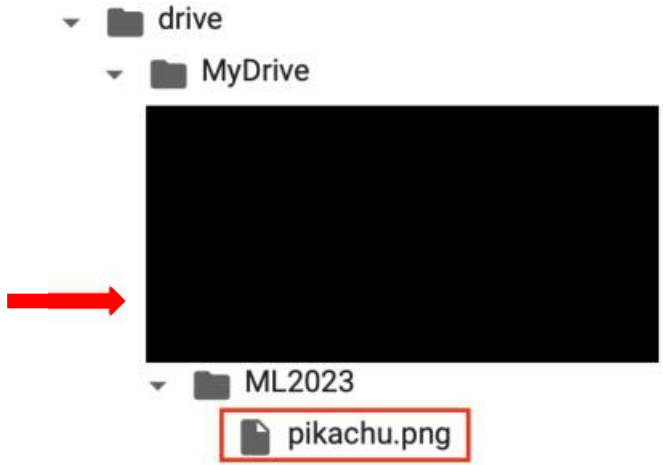
Execute the following three code blocks in order

This will download the image to your google drive, and you can access it later

```
[ ]  %cd /content/drive/MyDrive
     #change directory to google drive
     !mkdir ML2023 #make a directory named ML2023
     %cd ./ML2023
     #change directory to ML2023
```

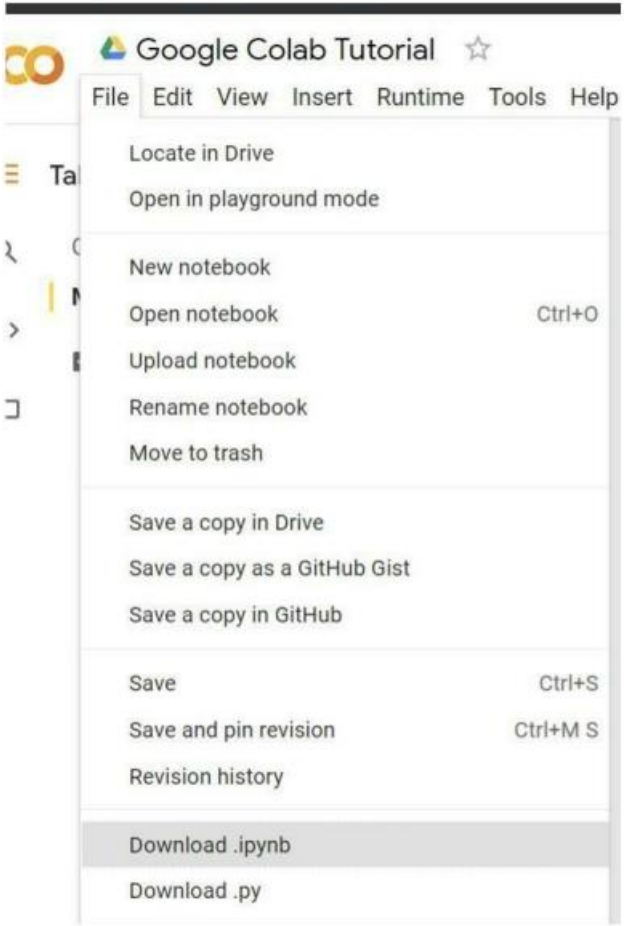
```
[ ]  !pwd #output the current directory
```

```
[ ]  !gdown --fuzzy https://drive.google.com/file/d/14FK5G6DOh7EdLyoj4D5teRSzriTOUPD7/view?usp=sharing --output pikachu.png
```



Saving Notebook

- Download the .ipynb file to your local device (File > Download .ipynb)
- Save the colab notebook to your google drive (File > Save a copy in Drive).
- Convert .ipynb to .py and download (File > Download .py)



Useful Linux Commands (in Colab)

ls : List all files in the current directory

ls -l : List all files in the current directory with more detail

pwd : Output the working directory

mkdir <dirname> : Create a directory <dirname>

cd <dirname> : Move to directory <dirname>

gdown : Download files from google drive

wget : Download files from the internet

python <python_file>: Executes a python file

Problems You May Encounter…

- Colab will **automatically disconnect** if idle timeout(90 min., sometimes varying) or when your screen goes black

  → solution: keep your screen on or try using javascript

- GPU usage is **not unlimited** ! (your account will be stopped for a period if you reached the max gpu usage 12 hrs)

  *** The cooldown period before you can connect to another GPU will extend from hours to days to weeks depending on your usage**

  → solution: open another account

Best solution:

1. buy colab pro :)
2. use your own resource (if able)

Reference

- https://colab.research.google.com
- https://research.google.com/colaboratory/faq.html