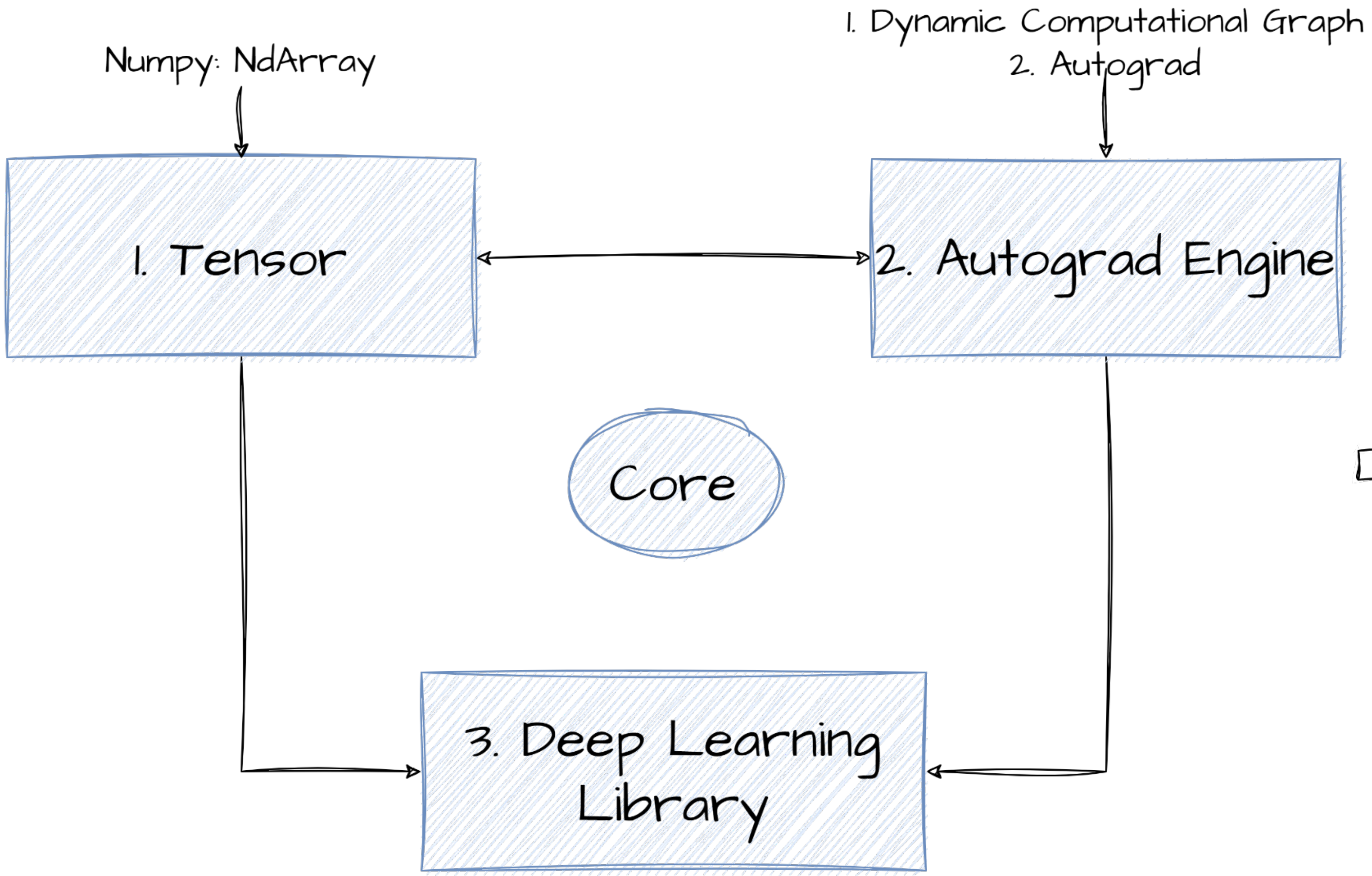
CB02-4 Pytorch Essentials

01.1 Essential Components of Pytorch



Numpy: NdArray

1. Dynamic Computational Graph
2. Autograd

1. Tensor

2. Autograd Engine

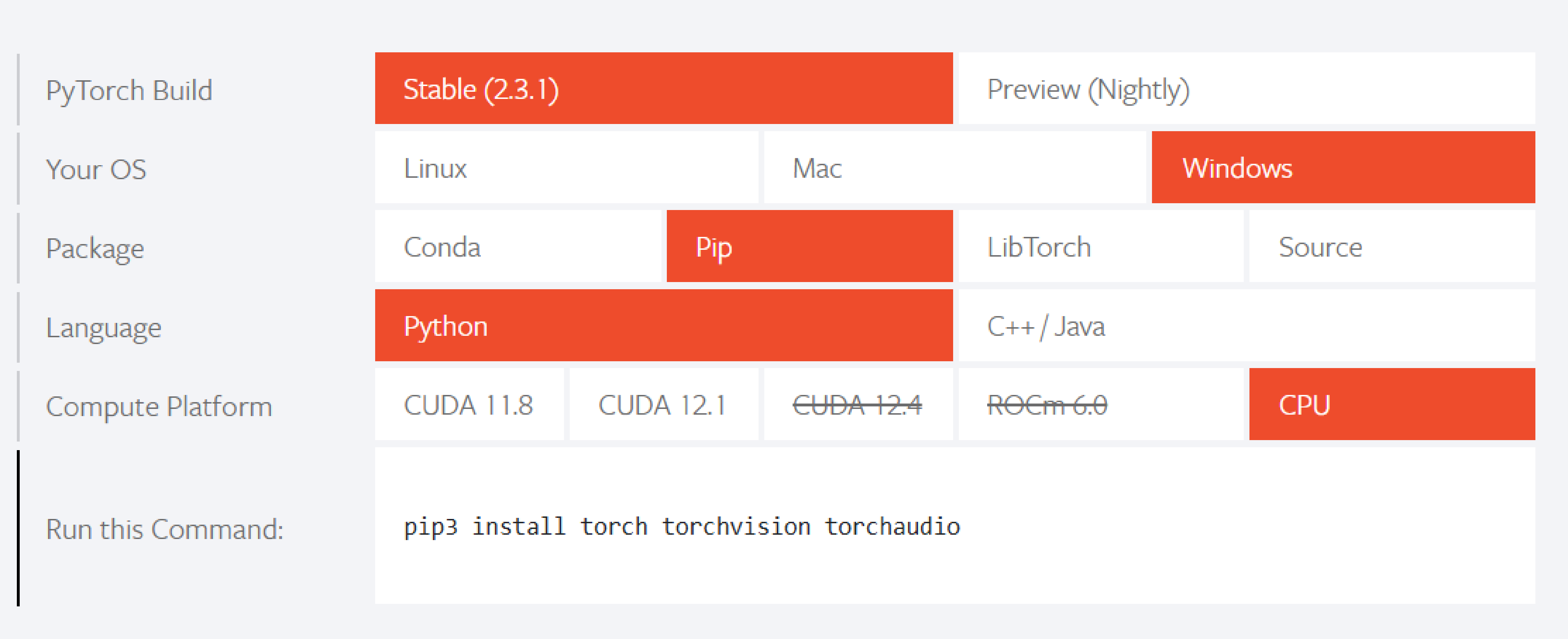
Core

3. Deep Learning Library

Other Components
4. 数据：dataset, dataloader
5. 算法：traning loop
6. 算力：GPU

01.2 Installation

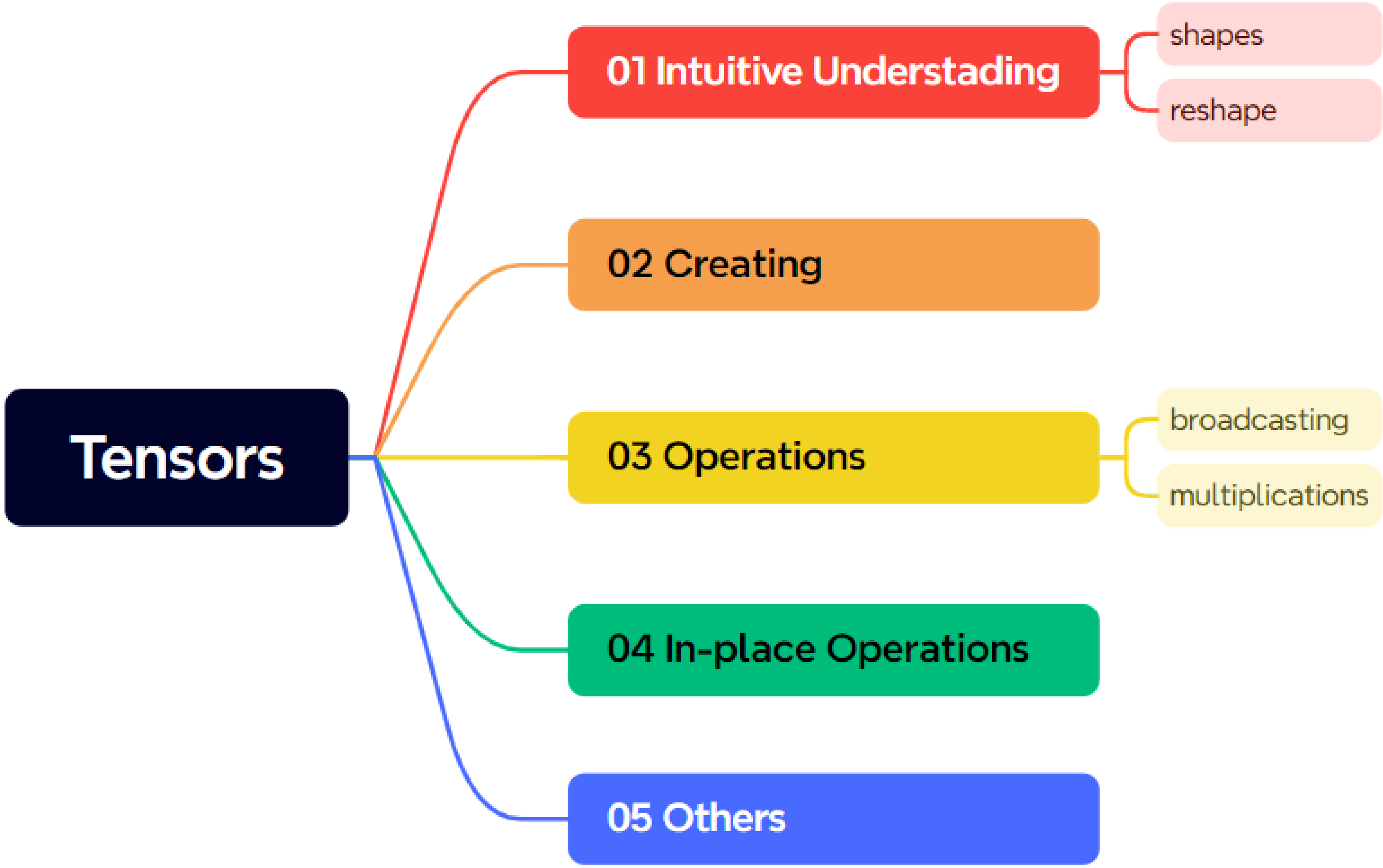
PyTorch: pytorch.org



PyTorch Build	[✓ Stable (2.3.1)]	Preview (Nightly)			
Your OS	Linux	Mac	[✓ Windows]		
Package	Conda	[✓ Pip]	LibTorch	Source	
Language	[✓ Python]	C++ / Java			
Compute Platform	CUDA 11.8	CUDA 12.1	~~CUDA 12.4~~	~~ROCm 6.0~~	[✓ CPU]
Run this Command:	`pip3 install torch torchvision torchaudio`				

```
pip3 install torch torchvision torchaudio
```

02 Tensors



Tensors

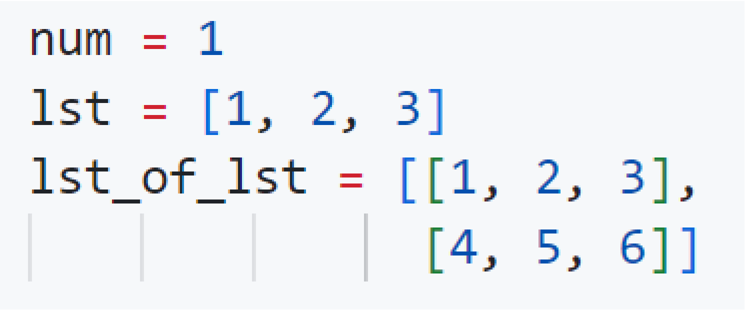
01 Intuitive Understading
- shapes
- reshape

02 Creating

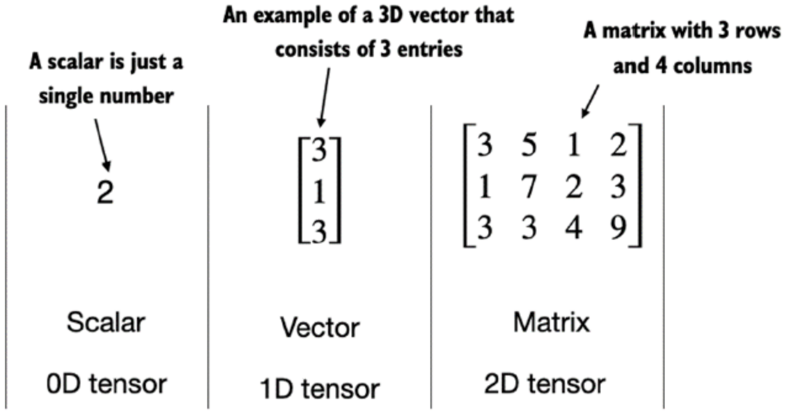
03 Operations
- broadcasting
- multiplications

04 In-place Operations

05 Others

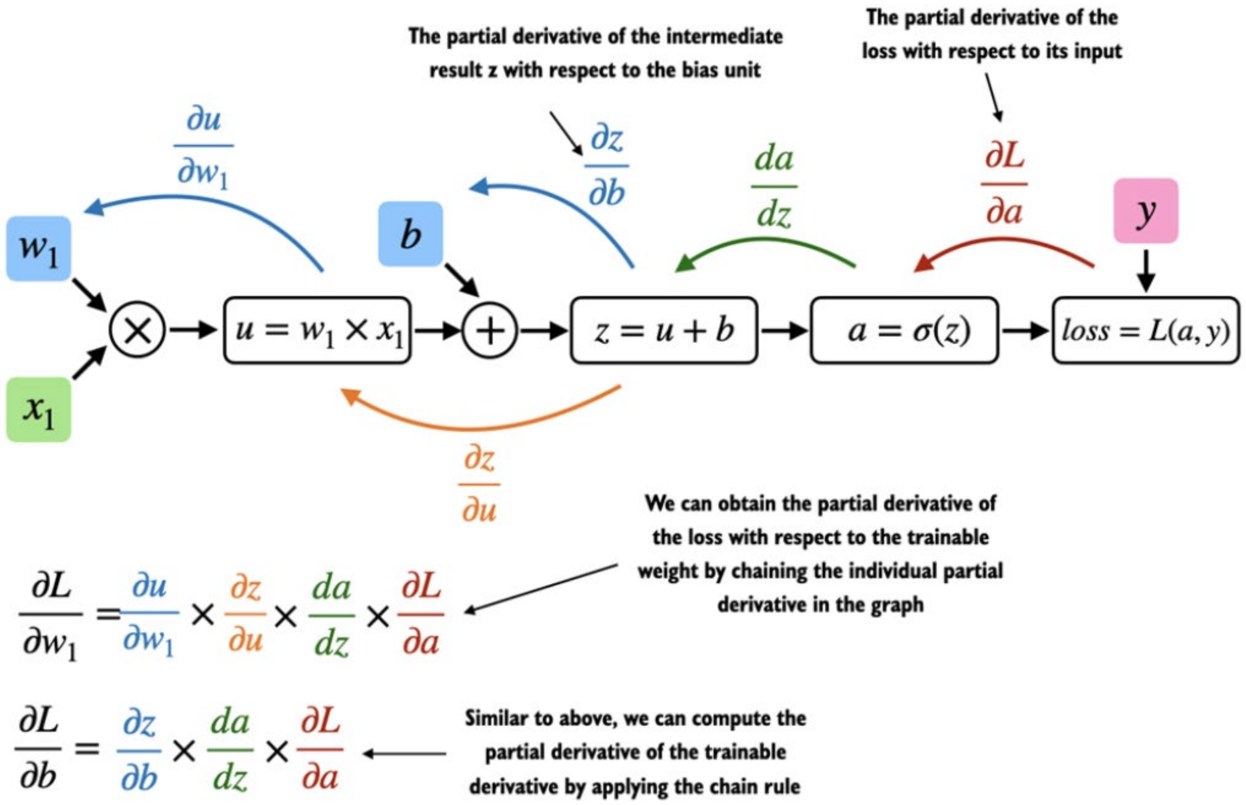


```
num = 1
lst = [1, 2, 3]
lst_of_lst = [[1, 2, 3],
              [4, 5, 6]]
```


A scalar is just a single number
2
Scalar
0D tensor
An example of a 3D vector that consists of 3 entries
3 1 3
Vector
1D tensor
A matrix with 3 rows and 4 columns
3 5 1 2
1 7 2 3
3 3 4 9
Matrix
2D tensor

Ref: Learn the Basics — PyTorch Tutorials 2.4.0+cu124 documentation

03 AutoGrad


w_1
x_1
b
y
$u = w_1 \times x_1$
$z = u + b$
$a = \sigma(z)$
$loss = L(a, y)$
$\frac{\partial u}{\partial w_1}$
$\frac{\partial z}{\partial b}$
$\frac{da}{dz}$
$\frac{\partial L}{\partial a}$
$\frac{\partial z}{\partial u}$

The partial derivative of the intermediate result z with respect to the bias unit

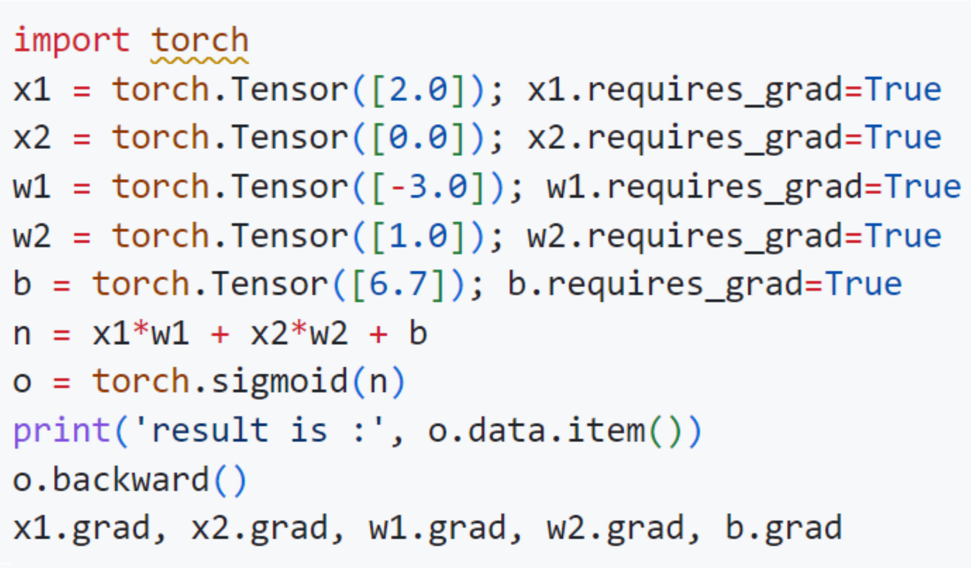
The partial derivative of the loss with respect to its input

$$\frac{\partial L}{\partial w_1} = \frac{\partial u}{\partial w_1} \times \frac{\partial z}{\partial u} \times \frac{da}{dz} \times \frac{\partial L}{\partial a}$$

We can obtain the partial derivative of the loss with respect to the trainable weight by chaining the individual partial derivative in the graph

$$\frac{\partial L}{\partial b} = \frac{\partial z}{\partial b} \times \frac{da}{dz} \times \frac{\partial L}{\partial a}$$

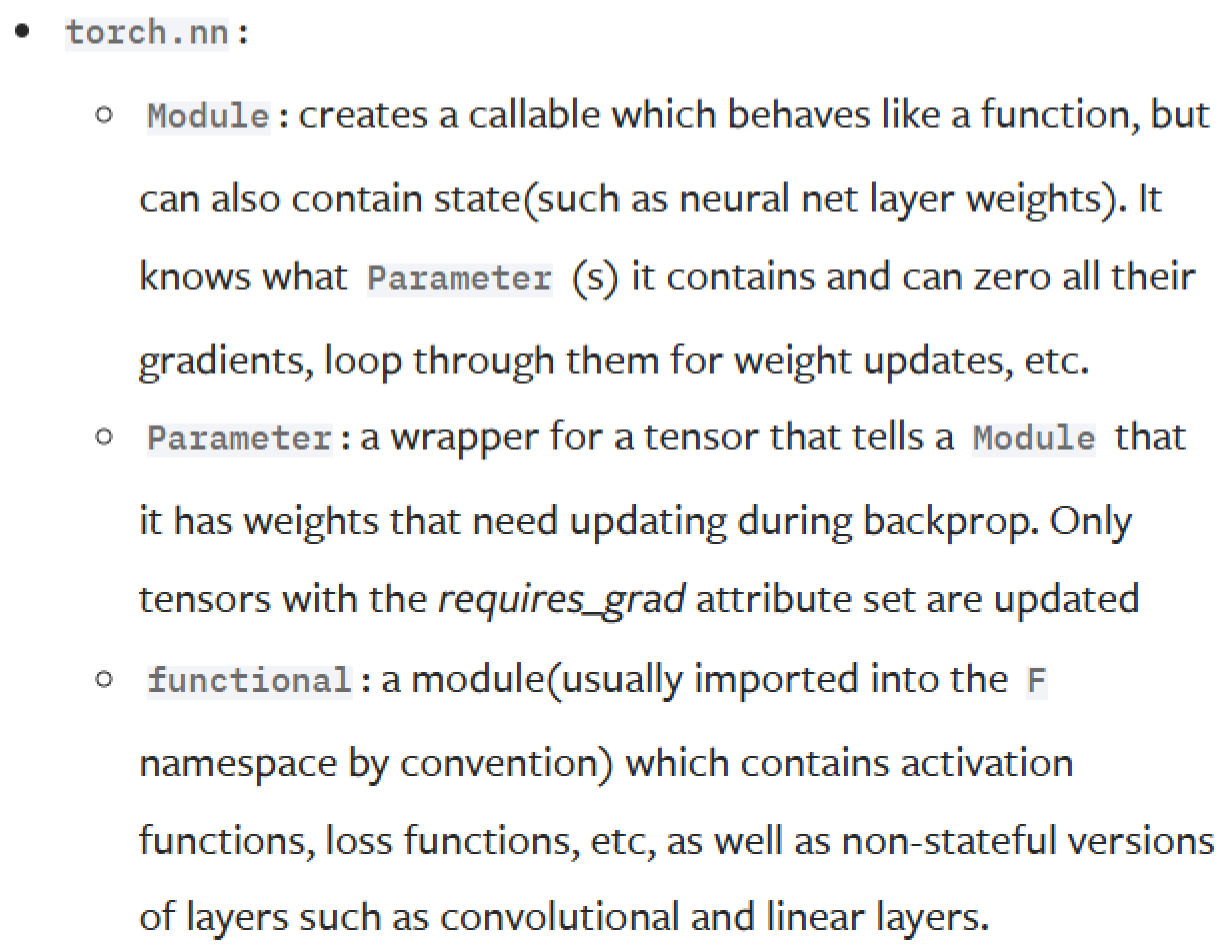
Similar to above, we can compute the partial derivative of the trainable derivative by applying the chain rule



```
import torch
x1 = torch.Tensor([2.0]); x1.requires_grad=True
x2 = torch.Tensor([0.0]); x2.requires_grad=True
w1 = torch.Tensor([-3.0]); w1.requires_grad=True
w2 = torch.Tensor([1.0]); w2.requires_grad=True
b = torch.Tensor([6.7]); b.requires_grad=True
n = x1*w1 + x2*w2 + b
o = torch.sigmoid(n)
print('result is :', o.data.item())
o.backward()
x1.grad, x2.grad, w1.grad, w2.grad, b.grad
```

Ref: Appendix A of ‘LLMs-from-scratch ‘(github.com)

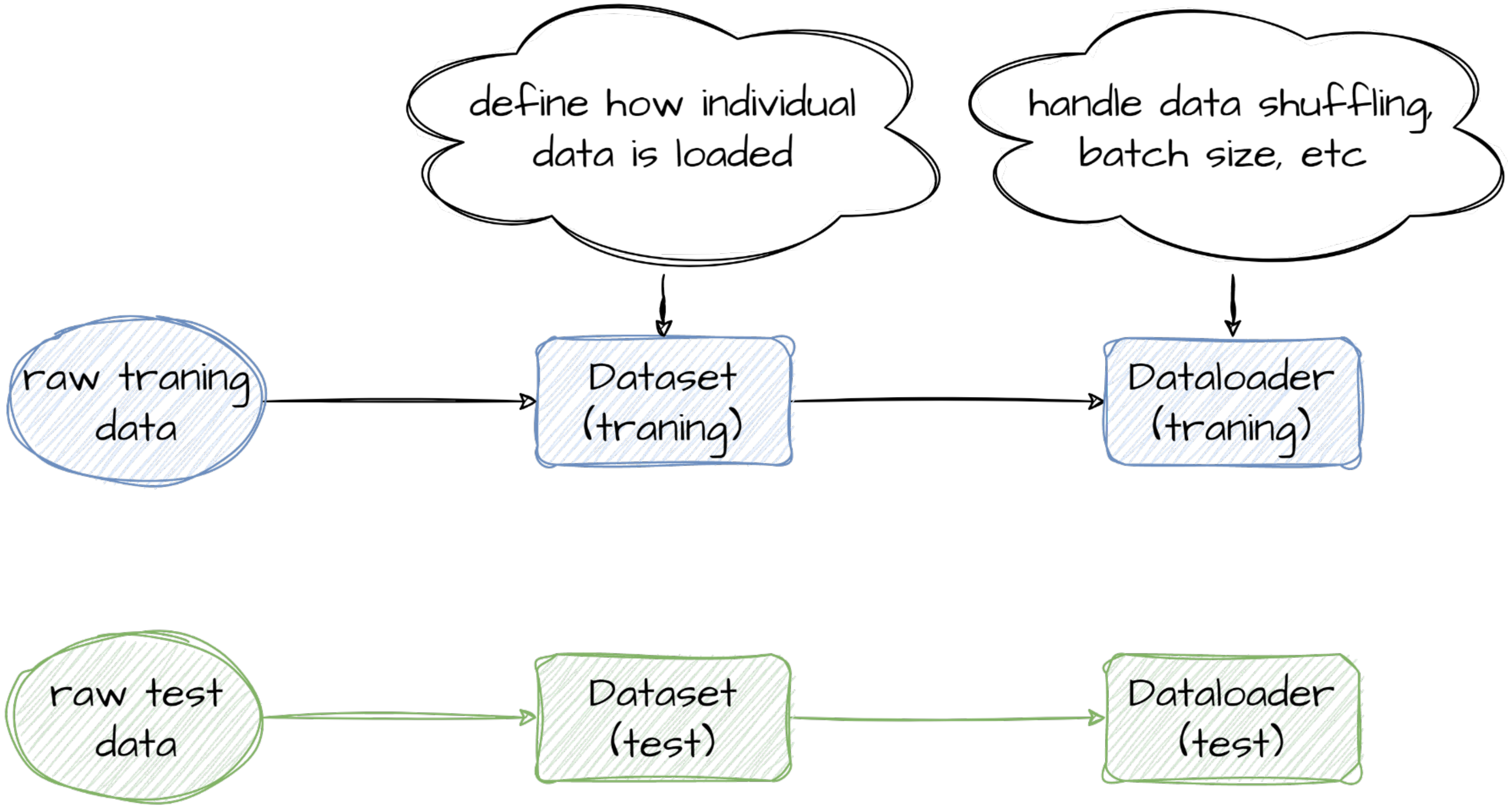
04 Building NN: torch.nn



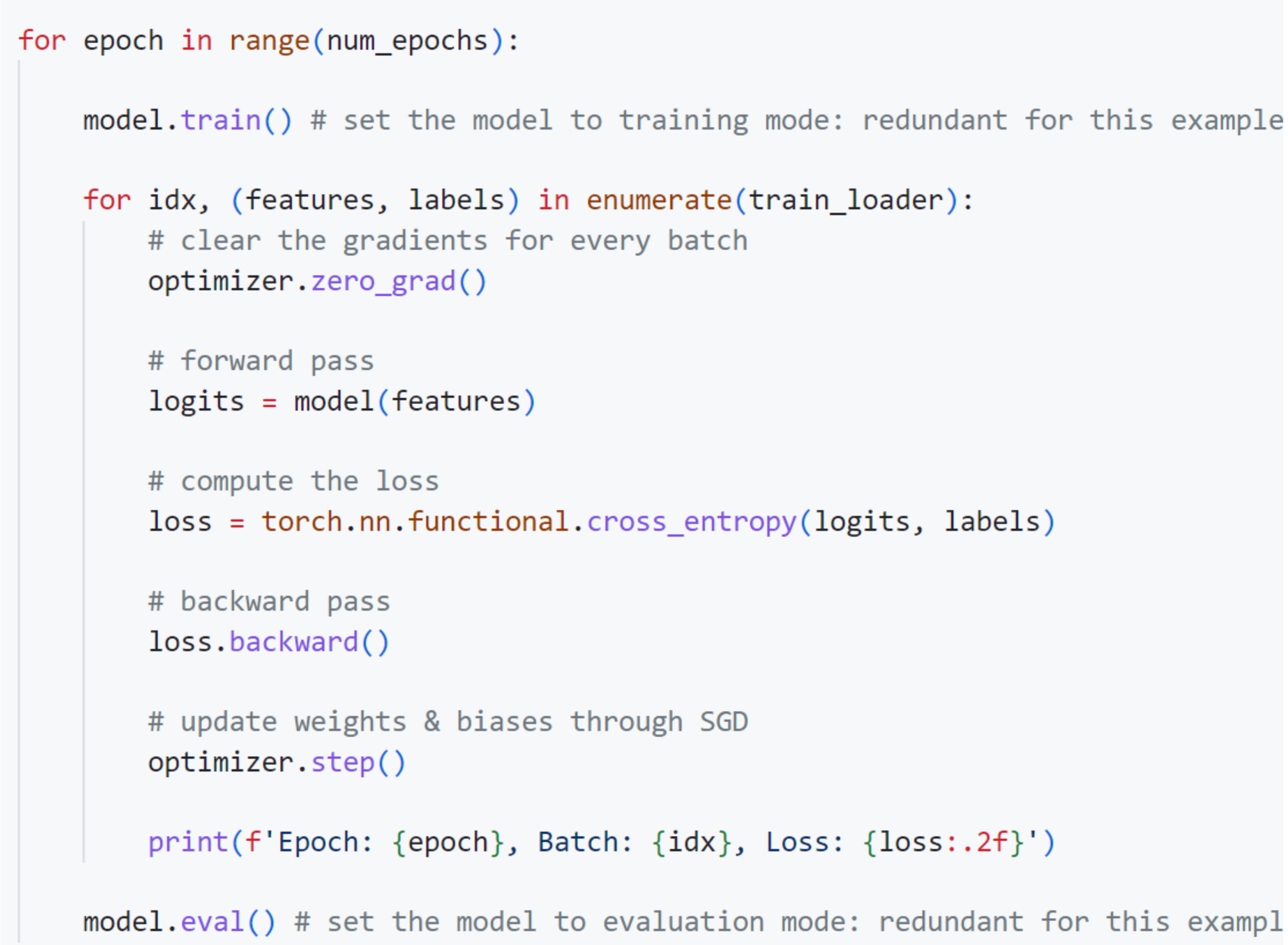
- `torch.nn`:
  - `Module`: creates a callable which behaves like a function, but can also contain state(such as neural net layer weights). It knows what `Parameter` (s) it contains and can zero all their gradients, loop through them for weight updates, etc.
  - `Parameter`: a wrapper for a tensor that tells a `Module` that it has weights that need updating during backprop. Only tensors with the *requires_grad* attribute set are updated
  - `functional`: a module(usually imported into the `F` namespace by convention) which contains activation functions, loss functions, etc, as well as non-stateful versions of layers such as convolutional and linear layers.

Ref: What is torch.nn really? — PyTorch Tutorials 2.4.0+cu124 documentation

05 Dataset & Dataloader



06 Typical Training Loop



```
for epoch in range(num_epochs):

    model.train() # set the model to training mode: redundant for this example

    for idx, (features, labels) in enumerate(train_loader):
        # clear the gradients for every batch
        optimizer.zero_grad()

        # forward pass
        logits = model(features)

        # compute the loss
        loss = torch.nn.functional.cross_entropy(logits, labels)

        # backward pass
        loss.backward()

        # update weights & biases through SGD
        optimizer.step()

        print(f'Epoch: {epoch}, Batch: {idx}, Loss: {loss:.2f}')

    model.eval() # set the model to evaluation mode: redundant for this exampl
```

Ref: cs230-stanford Code examples in pyTorch and Tensorflow for CS230 (github.com)

07 GPU

Single GPU: Pytorch GPU Version

Multiple GPUs:

1. accelerate(by HuggingFace)
2. DeepSpeed(by Microsoft)
3. DDP Module(by Pytorch)

GPT:

- 小型和中型项目、以及快速开发原型：倾向于使用Accelerate，因为它的简便性和与Hugging Face Transformers的兼容性。
- 大型项目和需要高性能优化的项目：通常选择DeepSpeed，因为它在大规模分布式训练中的表现非常优越，尤其是对于大模型和复杂训练任务。
- 高度自定义和灵活性要求高的项目：选择使用PyTorch的原生支持（如DDP），因为它提供了最大的灵活性和控制。

Other Tools: transformers, weights & biases(wnb)

Ref: DistributedDataParallel — PyTorch 2.3 documentation