按照Pytorch官方教程如何去构建一个神经网络模型

神经网络由对数据执行操作的层/模块组成。torch.nn 提供了构建自己的神经网络所需的所有构建块。PyTorch 中的每个模块都是nn.Module的子模块。一个神经网络本身是由其他模块（层）组成的模块。这种嵌套结构允许轻松构建和管理复杂的架构。

搭建步骤

导入模块

```
import os
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
```

Get Device for training

```
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using {device} device")
```

```
Using cuda device
```

定义网络架构

```
class NeuralNetwork(nn.Module): # 所有定义的网络都继承自nn.Module
    def __init__(self):
        super(NeuralNetwork, self).__init__() # 实例化调用父类__init__方法
        self.flatten = nn.Flatten() # 定义flatten
        self.linear_relu_stack = nn.Sequential( # 按照顺序去传入一些层
            # 5层
            nn.Linear(28*28, 512), # 线性层（MLP、前馈网络全连接层）,输入维度--->输出维度
            nn.ReLU(), # ReLu激活函数
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10), # 10分类
        )

    def forward(self, x): # forward代表对定义的网络的前向运算
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits # 得到10个类别的概率
```

```
# 实例化定义的网络
model = NeuralNetwork().to(device)
print(model)
```

```
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

使用

```
X = torch.rand(1, 28, 28, device=device) # 数据
logits = model(X)
pred_probab = nn.Softmax(dim=1)(logits) # 调用softmax，在dim=1维度上求和是等于1
y_pred = pred_probab.argmax(1) # 取最大值，即可以判断分类
print(f"Predicted class: {y_pred}")
```

```
Predicted class: tensor([0], device='cuda:0')
```

详细说明各网络模块

```
input_image = torch.rand(3,28,28)
print(input_image.size())
```

```
torch.Size([3, 28, 28])
```

nn.Flatten

```
flatten = nn.Flatten() # 从第一维到最后一维都变成一个维度 ---> (batch_size，其余维度相乘)
flat_image = flatten(input_image)
print(flat_image.size())
```

```
torch.Size([3, 784])
```

nn.Linear

```
layer1 = nn.Linear(in_features=28*28, out_features=20) # 输入维度 ---> 输出维度
hidden1 = layer1(flat_image)
print(hidden1.size())
```

```
torch.Size([3, 20])
```

nn.ReLu

```
print(f"Before ReLU: {hidden1}\n\n")
hidden1 = nn.ReLU()(hidden1)
print(f"After ReLU: {hidden1}")
```

```
Before ReLU: tensor([[-0.3503, -0.6784, -0.7309, -0.1807, -0.0332,  0.1426,
-0.1692, -0.1551,
          0.0252, -0.4042,  0.1829,  0.5812,  0.0019, -0.3465,  0.8111,  0.1311,
          0.4272, -0.2083, -0.0296, -0.1536],
        [-0.0380, -0.3993, -0.9975, -0.1506, -0.3178,  0.1786,  0.1844,  0.2409,
         -0.0206, -0.0528,  0.0540,  0.3856, -0.0977, -0.1008,  0.4515,  0.1772,
          0.1758,  0.1450, -0.1692, -0.4229],
        [-0.1344, -0.7771, -1.2344, -0.2608,  0.0700,  0.0703, -0.1827, -0.2296,
          0.2121, -0.3406, -0.2726,  0.4267, -0.1219, -0.1697,  0.5970,  0.3041,
          0.5153, -0.1691, -0.0183, -0.1043]], grad_fn=<AddmmBackward0>)
```

```
After ReLU: tensor([[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1426, 0.0000,
0.0000, 0.0252,
         0.0000, 0.1829, 0.5812, 0.0019, 0.0000, 0.8111, 0.1311, 0.4272, 0.0000,
         0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1786, 0.1844, 0.2409, 0.0000,
         0.0000, 0.0540, 0.3856, 0.0000, 0.0000, 0.4515, 0.1772, 0.1758, 0.1450,
         0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0700, 0.0703, 0.0000, 0.0000, 0.2121,
         0.0000, 0.0000, 0.4267, 0.0000, 0.0000, 0.5970, 0.3041, 0.5153, 0.0000,
         0.0000, 0.0000]], grad_fn=<ReluBackward0>)
```

nn.Sequential

```
# 按顺序把网络层串起来
seq_modules = nn.Sequential(
    flatten,
    layer1,
    nn.ReLU(),
    nn.Linear(20, 10)
)
input_image = torch.rand(3,28,28)
logits = seq_modules(input_image)
```

nn.Softmax

不同维度图解：https://zhuanlan.zhihu.com/p/525276061

```
softmax = nn.Softmax(dim=1) # 因为logits是二维的（batch_size,xxx），所以在dim=1上做
softmax
pred_probab = softmax(logits)
```

Model Parameters

```
# 打印模型参数
print(f"Model structure: {model}\n\n")

for name, param in model.named_parameters():
    print(f"Layer: {name} | Size: {param.size()} | Values : {param[:2]} \n")
```

```
Model structure: NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

```
Layer: linear_relu_stack.0.weight | Size: torch.Size([512, 784]) | Values :
tensor([[ 0.0231, -0.0284,  0.0133,  ..., -0.0106,  0.0142, -0.0097],
        [-0.0246,  0.0179, -0.0065,  ..., -0.0235,  0.0060, -0.0346]],
       device='cuda:0', grad_fn=<SliceBackward0>)

Layer: linear_relu_stack.0.bias | Size: torch.Size([512]) | Values :
tensor([-0.0162,  0.0317], device='cuda:0', grad_fn=<SliceBackward0>)

Layer: linear_relu_stack.2.weight | Size: torch.Size([512, 512]) | Values :
tensor([[ 0.0127, -0.0062,  0.0040,  ...,  0.0087,  0.0375,  0.0125],
        [ 0.0069,  0.0087,  0.0217,  ...,  0.0059, -0.0143, -0.0161]],
       device='cuda:0', grad_fn=<SliceBackward0>)

Layer: linear_relu_stack.2.bias | Size: torch.Size([512]) | Values :
tensor([0.0037, 0.0086], device='cuda:0', grad_fn=<SliceBackward0>)

Layer: linear_relu_stack.4.weight | Size: torch.Size([10, 512]) | Values :
tensor([[ 0.0236,  0.0215, -0.0347,  ...,  0.0272, -0.0419,  0.0289],
        [-0.0438,  0.0243,  0.0198,  ..., -0.0421, -0.0375, -0.0175]],
       device='cuda:0', grad_fn=<SliceBackward0>)

Layer: linear_relu_stack.4.bias | Size: torch.Size([10]) | Values : tensor([
3.4353e-02, -5.4434e-05], device='cuda:0', grad_fn=<SliceBackward0>)
```