5. 模型定义

5.1 pytorch定义模型

1. nn.Sequential()

1. key-value形式-字典形式

```
# -*- coding: utf-8 -*-
# Date: 2022/3/15 0015
# Time: 11:46
# Author: HQR

import torch.nn as nn
import torch
from collections import OrderedDict
class MySequential(nn.Module):
    def __init__(self, *args):
        super(MySequential, self).__init__()
        if len(args) == 1 and isinstance(args[0], OrderedDict): # 如果传入的是一个
OrderedDict
            for key, module in args[0].items():
                self.add_module(key, module)  # add_module方法会将module添加进
self._modules(一个OrderedDict)
        else:  # 传入的是一些Module
            for idx, module in enumerate(args):
                self.add_module(str(idx), module)
    def forward(self, input):
        # self._modules返回一个 OrderedDict，保证会按照成员添加时的顺序遍历成
        for module in self._modules.values():
            input = module(input)
        return input

if __name__ == '__main__':
    arg = OrderedDict([
        ('conv1', nn.Conv2d(3, 60, (3, 3))),
        ('relu1', nn.ReLU()),
        ('fc1', nn.Conv2d(60, 128, (3, 3))),
        ('relu2', nn.ReLU())
    ])
    model = MySequential(arg)
    input = torch.ones(1, 3, 224, 224)
    print(model)
    print(model(input).shape)
```

```
D:\HQR_Anaconda\envs\tf-gpu\python.exe "D:/Python/PyCharmProjects/Datawhale_pytorch/Chapter5/1. test_Sequential.py"
MySequential(
  (conv1): Conv2d(3, 60, kernel_size=(3, 3), stride=(1, 1))
  (relu1): ReLU()
  (fc1): Conv2d(60, 128, kernel_size=(3, 3), stride=(1, 1))
  (relu2): ReLU()
)
torch.Size([1, 128, 220, 220])
```

2. 直排

```python
# -*- coding: utf-8 -*-
# Date: 2022/3/16 0016
# Time: 21:05
# Author: HQR

import torch
from torch import nn
from collections import OrderedDict


class MySequential(nn.Module):
    def __init__(self, *args):
        super(MySequential, self).__init__()
        if len(args) == 1 and isinstance(args[0], OrderedDict): # 如果传入的是一个
OrderedDict
            for key, module in args[0].items():
                self.add_module(key, module)  # add_module方法会将module添加进
self._modules(一个OrderedDict)
        else:  # 传入的是一些Module
            for idx, module in enumerate(args):
                self.add_module(str(idx), module)
    def forward(self, x):
        # self._modules返回一个 OrderedDict，保证会按照成员添加时的顺序遍历成
        for module in self._modules.values():
            # self._modules.values 保存的是key-value
            print(module)
            x = module(x)
        return x


if __name__ == '__main__':
    args = nn.Sequential(
        nn.Linear(128, 256),
        nn.ReLU(),
        nn.Linear(256, 5),
    )
    model = MySequential(args)
    x = torch.randn(1,128)
    print(model(x).shape)
```

```
D:\HQR_Anaconda\envs\tf-gpu\python.exe "D:/Python/PyCharmProjects/Datawhale_pytorch/Chapter5/2. line_Sequential.py"
Sequential(
  (0): Linear(in_features=128, out_features=256, bias=True)
  (1): ReLU()
  (2): Linear(in_features=256, out_features=5, bias=True)
)
torch.Size([1, 5])

Process finished with exit code 0
```

2. nn.ModuleList

对应模块为nn.ModuleList()。

- ModuleList 接收一个子模块（或层，需属于nn.Module类）的列表作为输入，然后也可以**类似List那样进行append和extend操作**。同时，子模块或层的权重也会自动添加到网络中来。
- nn.ModuleList 并没有定义一个网络，它只是将不同的模块储存在一起。ModuleList中元素的先后顺序并不代表其在网络中的真实位置顺序，需要经过forward函数指定各个层的先后顺序后才算完成了模型的定义。具体实现时用for循环即可完成：

```python
# -*- coding: utf-8 -*-
# Date: 2022/3/16 0016
# Time: 21:14
# Author: HQR
import torch
from torch import nn


class model(nn.Module):
    def __init__(self, args):
        super(model, self).__init__()
        self.ModuleList = args

    def forward(self, x):
        for m in self.ModuleList:
            x = m(x)
        return x


if __name__ == '__main__':
    args = nn.ModuleList([
        nn.Linear(784, 256),
        nn.ReLU(),
        nn.Linear(256, 10)
    ])
    net = model(args)
    print(net)
    # model(
    #   (modules): ModuleList(
    #     (0): Linear(in_features=10, out_features=10, bias=True)
    #     (1): ReLU()
    #     (2): Linear(in_features=10, out_features=10, bias=True)
    #   )
    # )

    for param in net.parameters():
        print(type(param.data), param.size())

    # class 'torch.Tensor'> torch.Size([256, 784])
    # class 'torch.Tensor'> torch.Size([256])
    # class 'torch.Tensor'> torch.Size([10, 256])
    # class 'torch.Tensor'> torch.Size([10])
    x = torch.randn(1, 784)
    print(net)
    print(net(x).shape)
```

```
<class 'torch.Tensor'> torch.Size([256, 784])
<class 'torch.Tensor'> torch.Size([256])
<class 'torch.Tensor'> torch.Size([10, 256])
<class 'torch.Tensor'> torch.Size([10])
model(
  (ModuleList): ModuleList(
    (0): Linear(in_features=784, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=10, bias=True)
  )
)
torch.Size([1, 10])
```

3. nn.ModuleDict

```
import torch.nn as nn
net = nn.ModuleDict({
    'linear': nn.Linear(784, 256),
    'act': nn.ReLU(),
})
net['output'] = nn.Linear(256, 10) # 添加
print(net['linear']) # 访问
print(net.output)
print(net)
```

```
Linear(in_features=784, out_features=256, bias=True)
Linear(in_features=256, out_features=10, bias=True)
ModuleDict(
  (linear): Linear(in_features=784, out_features=256, bias=True)
  (act): ReLU()
  (output): Linear(in_features=256, out_features=10, bias=True)
)
```

总结

- Sequential适用于快速验证结果，因为已经明确了要用哪些层，直接写一下就好了，不需要同时写**init**和forward；
- ModuleList和ModuleDict在某个完全相同的层需要重复出现多次时，非常方便实现，可以”一行顶多行“；
- 当我们需要之前层的信息的时候，比如 ResNets 中的残差计算，当前层的结果需要和之前层中的结果进行融合，一般使用 ModuleList/ModuleDict 比较方便。