以VGG为例：

1. print

```
import torch
from torch import nn
from collections import OrderedDict
import torch.nn.functional as F
import torchvision

model = torchvision.models.vgg11()
print("model: ",model)
"""
model:  VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (11): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): ReLU(inplace=True)
    (13): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (14): ReLU(inplace=True)
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (16): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(inplace=True)
    (18): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (19): ReLU(inplace=True)
    (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
"""
```

print输出的是一个类，<class 'torchvision.models.vgg.VGG'>

2. parameters() 或者 named_parameters方法来访问所有参数

```
model = nn.Sequential(nn.Linear(4, 3), nn.ReLU(), nn.Linear(3, 1))
print("model: ",model,type(model))
"""
model:  Sequential(
  (0): Linear(in_features=4, out_features=3, bias=True)
  (1): ReLU()
  (2): Linear(in_features=3, out_features=1, bias=True)
) <class 'torch.nn.modules.container.Sequential'>
"""

print(type(model.parameters()),type(model.named_parameters()))
# <class 'generator'> <class 'generator'>

pa = [i for i in model.parameters()]
print(pa)
'''
[Parameter containing:
tensor([[-0.2491, -0.2851,  0.4488,  0.0664],
        [ 0.0551,  0.1467,  0.0895,  0.4776],
        [-0.2417,  0.1569, -0.0995, -0.0746]], requires_grad=True), Parameter containing:
tensor([-0.2431,  0.4721,  0.2136], requires_grad=True), Parameter containing:
tensor([[ 0.4174, -0.4439, -0.4951]], requires_grad=True), Parameter containing:
tensor([0.5765], requires_grad=True)]
'''
print(pa[0],type(pa[0],pa[0].size()))
"""
Parameter containing:
tensor([[ 0.4244,  0.4701, -0.4603,  0.0722],
        [ 0.3741,  0.2089,  0.1482, -0.4755],
        [-0.2416,  0.1858,  0.3110,  0.2068]], requires_grad=True) <class 'torch.nn.parameter.P
torch.Size([3, 4])
"""

for named,param in model.named_parameters():
  print(named,param.size())
"""
0.weight torch.Size([3, 4])
0.bias torch.Size([3])
2.weight torch.Size([1, 3])
2.bias torch.Size([1])
"""
```

3. torch.nn.parameter.Parameter 是Tensor的子类

和Tensor不同的是如果一个Tensor是Parameter，那么它会自动被添加到模型的参数列表里，来看下面这个例子。

```
class MyModel(nn.Module):
    def __init__(self, **kwargs):
        super(MyModel, self).__init__(**kwargs)
        self.weight1 = nn.Parameter(torch.rand(20, 20))
        self.weight2 = torch.rand(20, 20)
    def forward(self, x):
        pass

n = MyModel()
for name, param in n.named_parameters():
    print(name)
# weight1
```

上面的代码中weight1在参数列表中但是weight2却没在参数列表中。

因为Parameter是Tensor，即Tensor拥有的属性它都有，比如可以根据data来访问参数数值，用grad来访问参数梯度。

```
weight_0 = list(net[0].parameters())[0]
print(weight_0.data)
print(weight_0.grad) # 反向传播前梯度为None
Y.backward()
print(weight_0.grad)
"""
tensor([[ 0.2719, -0.0898, -0.2462,  0.0655],
        [-0.4669, -0.2703,  0.3230,  0.2067],
        [-0.2708,  0.1171, -0.0995,  0.3913]])
None
tensor([[-0.2281, -0.0653, -0.1646, -0.2569],
        [-0.1916, -0.0549, -0.1382, -0.2158],
        [ 0.0000,  0.0000,  0.0000,  0.0000]])

"""
```

4. model.named_patameters(),model.parameters(),model.state_dict().items()

```
model = nn.Sequential(nn.Linear(4, 3), nn.ReLU(), nn.Linear(3, 1))
print(model.state_dict(),type(model.state_dict()))
"""
OrderedDict([('0.weight', tensor([[ 0.4855,  0.4483, -0.4869, -0.0463],
        [-0.1612,  0.1065, -0.0085,  0.4753],
        [ 0.0390, -0.3921, -0.4056, -0.1686]])), ('0.bias', tensor([-0.3444, -0.3594, -0.4876]
"""

for name,param in model.state_dict().items():
    print(name,param) #,param.requires_grad=True)
"""
0.weight tensor([[ 0.2481,  0.0799, -0.4800,  0.0551],
        [ 0.4119,  0.2315,  0.2823,  0.2748],
        [ 0.0974,  0.2094, -0.3300,  0.0290]])
0.bias tensor([-0.2839,  0.1462,  0.0261])
2.weight tensor([[ 0.0612, -0.3314,  0.0713]])
2.bias tensor([-0.2062])
"""
```

> 1. model.named_parameters()，迭代打印model.named_parameters()将会打印每一次迭代元素的名字和param
> 2. model.parameters()，迭代打印model.parameters()将会打印每一次迭代元素的param而不会打印名字，这是他和named_parameters的区别，两者都可以用来改变requires_grad的属性
> 3. model.state_dict().items() 每次迭代打印该选项的话，会打印所有的name和param，但是这里的所有的param都是requires_grad=False,没有办法改变requires_grad的属性，所以改变requires_grad的属性只能通过上面的两种方式。
> 4. 改变了requires_grad之后要修改optimizer的属性
>   optimizer = optim.SGD(
>   filter(lambda p: p.requires_grad, model.parameters()), #只更新requires_grad=True的参数
>   lr=cfg.TRAIN.LR,
>   momentum=cfg.TRAIN.MOMENTUM,
>   weight_decay=cfg.TRAIN.WD,
>   nesterov=cfg.TRAIN.NESTEROV
>   )