Python小练习：权重初始化（Weight Initialization）

作者：凯鲁嘎吉 - 博客园 http://www.cnblogs.com/kailugaji/

调用Pytorch中的torch.nn.init.xxx实现对模型权重与偏置初始化。

1. weight_init_test.py

```python
# -*- coding: utf-8 -*-
# Author：凯鲁嘎吉 Coral Gajic
# https://www.cnblogs.com/kailugaji/
# Python小练习：权重初始化（Weight Initialization）
# Custom weight init for Conv2D and Linear layers.
import torch
import torch.nn.functional as F
import torch.nn as nn
# 根据网络层的不同定义不同的初始化方式
# 以下是两种不同的初始化方式：
# 正态分布+常数
def weight_init(m):
    if isinstance(m, nn.Linear):
        # 如果传入的参数是 nn.Linear 类型，则执行以下操作：
        nn.init.xavier_normal_(m.weight) # 将权重初始化为 Xavier 正态分布
        nn.init.constant_(m.bias, 0) # 将权重初始化为常数
    elif isinstance(m, nn.Conv2d):
        # 如果传入的参数是 nn.Conv2d 类型，则执行以下操作：
        nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu') # 将权重初始化为正态分布
    elif isinstance(m, nn.BatchNorm2d):
        # 如果传入的参数是 nn.BatchNorm2d 类型，则执行以下操作：
        nn.init.constant_(m.weight, 1)
        nn.init.constant_(m.bias, 0)

# 正交+常数
def weight_init2(m):
    if isinstance(m, nn.Linear):
        # 如果传入的参数是 nn.Linear 类型，则执行以下操作：
        nn.init.orthogonal_(m.weight.data) # 对权重矩阵进行正交化操作，使其具有对称性。
        if hasattr(m.bias, 'data'):
            m.bias.data.fill_(0.0) # 如果传入的参数包含偏置项，则将其填充为零。
    elif isinstance(m, nn.Conv2d) or isinstance(m, nn.ConvTranspose2d):
        # 如果传入的参数是 nn.Conv2d 或 nn.ConvTranspose2d 类型，则执行以下操作：
```

```
        gain = nn.init.calculate_gain('relu') # 用于计算激活函数的增益
        nn.init.orthogonal_(m.weight.data, gain) # 对权重矩阵进行正交化操作，使其具有对称性。
        if hasattr(m.bias, 'data'):
            m.bias.data.fill_(0.0) # 如果传入的参数包含偏置项，则将其填充为零。

class Net(nn.Module):
    def __init__(self, input_size=1):
        self.input_size = input_size
        super(Net, self).__init__()
        self.fc1 = nn.Linear(self.input_size, 2)
        self.fc2 = nn.Linear(2, 4)
        self.fc3 = nn.Linear(4, 2)

    def forward(self, x):
        x = x.view(-1, self.input_size)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)

torch.manual_seed(1)
num = 4 # 输入维度
x = torch.randn(1, num)
# 方式1：
model = Net(input_size = num)
print('网络结构：\n', model)
print('输入：\n', x)
model.apply(weight_init)
y = model(x)
print('输出1：\n', y.data)
print('权重1：\n', model.fc1.weight.data)
# 方式2：
model = Net(input_size = num)
model.apply(weight_init2)
y = model(x)
print('输出2：\n', y.data)
print('权重2：\n', model.fc1.weight.data)
```

2. 结果

```
D:\ProgramData\Anaconda3\python.exe "D:/Python code/2023.3 exercise/Neural Network/weight_init_test.py"
网络结构：
 Net(
  (fc1): Linear(in_features=4, out_features=2, bias=True)
  (fc2): Linear(in_features=2, out_features=4, bias=True)
```

```
  (fc3): Linear(in_features=4, out_features=2, bias=True)
)
输入：
 tensor([[0.6614, 0.2669, 0.0617, 0.6213]])
输出1：
 tensor([[-0.7233, -0.6639]])
权重1：
 tensor([[ 2.0709, -1.0573,  0.9230, -0.7373],
        [ 0.1879, -0.2766,  0.7962,  1.4599]])
输出2：
 tensor([[-0.6951, -0.6912]])
权重2：
 tensor([[-0.8471, -0.4721,  0.1653,  0.1795],
        [-0.4072,  0.5991, -0.6437,  0.2467]])

Process finished with exit code 0
```

完成。