**多层感知器PyTorch实现二**

上期文章我们基本上是一步步来实现我们的多层感知机模型的：

**多层感知器PyTorch实现一**

接下来我们将使用Pytorch中的相关API进行实现，这将大大缩短我们实现的代码量，接下来我们的实现如下所示：

1. **数据加载**

数据加载的代码跟之前的一样，我们这里直接使用之前的代码实现：

def load\_data\_fashion\_mnist(batch\_size, resize=None):  
 root = "E:\DataSet\DLPytorch\Data\FashionMNIST"  
 trans = [transforms.ToTensor()]  
 if resize:  
 trans.insert(0, transforms.Resize(resize))  
 trans = transforms.Compose(trans)  
 fmnist\_train = torchvision.datasets.FashionMNIST(root=root, train=True, transform=trans, download=True)  
 fmnist\_test = torchvision.datasets.FashionMNIST(root=root, train=False, transform=trans, download=True)  
 return (data.DataLoader(fmnist\_train, batch\_size, shuffle=True),  
 data.DataLoader(fmnist\_test, batch\_size, shuffle=False))

1. **定义网络结构**

接下来是定义网络结构，定义网络结构的时候我们使用Sequence模型结构，基于torch.nn的模块来实现：

*# 定义网络结构*net = nn.Sequential(  
 *# Flatten是展平层，返回一个一维数组,来调整网络输入的形状* nn.Flatten(),  
 *# 第一层是隐含层，包含784个输入单元，256个隐藏单元* nn.Linear(784,256),  
 *# 对第一层的输入进行ReLU激活* nn.ReLU(),  
 *# 输出层，输入维度是256，输出维度为10* nn.Linear(256,10)  
)

这里我们打印一下网络结构如下所示：

*# 输出网络结构*print(net)

Sequential(

(0): Flatten(start\_dim=1, end\_dim=-1)

(1): Linear(in\_features=784, out\_features=256, bias=True)

(2): ReLU()

(3): Linear(in\_features=256, out\_features=10, bias=True)

)

基于此，我们可以看出我们的网络结构还是很简单的，在后面的教程中我们将可视化我们的网络结构，这里先不进行阐述。

**三、定义初始化模型参数**

这里的模型参数我们使用以下方法进行构建：

def init\_weights(m):  
 if type(m) == nn.Linear:  
 nn.init.normal\_(m.weight, std=0.01)  
 nn.init.zeros\_(m.bias)  
net.apply(init\_weights)  
print(net.parameters())

# 输出

<generator object Module.parameters at 0x000001D006D5EA50>

可以看出这个参数是一个generator 对象，即生成器对象，这里我们使用for循环来对初始化的参数进行查看：

for parm in net.parameters():  
 print(parm)

# 输出：

Parameter containing:

tensor([[ 0.0069, 0.0060, 0.0049, ..., 0.0155, -0.0217, -0.0093],

[ 0.0074, -0.0282, 0.0121, ..., -0.0052, 0.0155, 0.0051],

[ 0.0037, 0.0010, -0.0160, ..., 0.0031, -0.0040, -0.0087],

...,

[-0.0005, -0.0057, 0.0067, ..., -0.0108, 0.0088, -0.0069],

[-0.0073, 0.0004, 0.0156, ..., 0.0078, 0.0182, 0.0051],

[-0.0034, -0.0229, -0.0085, ..., 0.0114, 0.0139, -0.0066]],

requires\_grad=True)

Parameter containing:

tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],

requires\_grad=True)

Parameter containing:

tensor([[-0.0051, -0.0006, 0.0141, ..., -0.0009, 0.0011, 0.0090],

[-0.0035, 0.0009, 0.0050, ..., -0.0042, 0.0100, -0.0186],

[ 0.0053, 0.0068, 0.0075, ..., 0.0022, 0.0019, -0.0054],

...,

[ 0.0168, -0.0003, 0.0104, ..., -0.0040, 0.0082, 0.0084],

[ 0.0131, 0.0009, -0.0085, ..., -0.0274, -0.0021, -0.0230],

[-0.0099, 0.0214, 0.0109, ..., -0.0082, 0.0006, 0.0056]],

requires\_grad=True)

Parameter containing:

tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], requires\_grad=True)

稍微解释一下这里的输出，由于我们输入层到隐藏层有参数，隐藏层到输出层也是有参数的，均还有w和b参数。Parameter containing后的参数均是w和b初始化的参数，

**四、损失函数及优化算法**

损失函数和优化算法我们参考之前的实现即可：

batch\_size,num\_epochs,lr = 256,10,0.1  
loss = nn.CrossEntropyLoss()  
*# 定义优化算法为随机梯度下降  
# 参数使用优化的参数即可*optimizer = torch.optim.SGD(net.parameters(),lr = lr)  
print(optimizer)

**# 输出**

SGD (

Parameter Group 0

dampening: 0

lr: 0.1

momentum: 0

nesterov: False

weight\_decay: 0

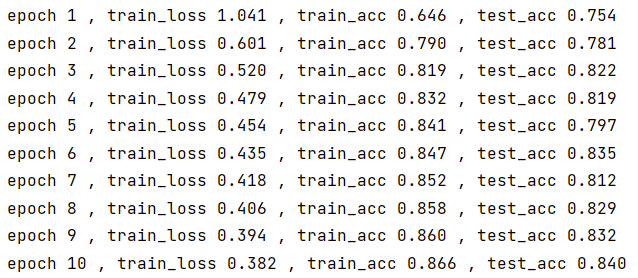
)

1. **模型训练**

模型训练的代码我们可以直接参考之前的实现：

class Accumulator:  
 def \_\_init\_\_(self, n):  
 self.data = [0.0] \* n  
 *# 进行累加* def add(self, \*args):  
 self.data = [a + float(b) for a, b in zip(self.data, args)]  
 def reset(self):  
 self.data = [0.0] \* len(self.data)  
 def \_\_getitem\_\_(self, idx):  
 return self.data[idx]  
  
def accuracy(y\_hat,y):  
 if len(y\_hat.shape)>1 and y\_hat.shape[1]>1:  
 *# 寻找哪一项是最大的* y\_hat = y\_hat.argmax(axis =1)  
 cmp = y\_hat.type(y.dtype) == y  
 *# print('CMP:',cmp)* return float(cmp.type(y.dtype).sum())  
  
def evaluate\_accuracy(net, data\_iter):  
 *# 将模型设置为评估模式* if isinstance(net, torch.nn.Module):  
 net.eval()  
 *# 预测正确的数量和预测数量* metric = Accumulator(2) *# No. of correct predictions, no. of predictions* with torch.no\_grad():  
 for X, y in data\_iter:  
 metric.add(accuracy(net(X), y), y.numel())  
 *# 精度* return metric[0] / metric[1]  
  
  
def train(net, train\_data, loss, optimizer):  
 *# 将模型设置为训练模式* if isinstance(net, torch.nn.Module):  
 net.train()  
 *# 累加训练损失值、累加训练精度，统计样本数  
 # 初始化* metric = Accumulator(3)  
 for X, y in train\_data:  
 *# 计算梯度和更新参数* y\_hat = net(X)  
 *# 样本损失* l = loss(y\_hat, y)  
 if isinstance(optimizer, torch.optim.Optimizer):  
 *# 梯度清零，即将优化器的梯度清零，避免梯度累加的影响* optimizer.zero\_grad()  
 *# 利用反向传播算法计算损失函数关于模型参数的梯度。* l.mean().backward()  
 *# l.sum().backward()  
 # 利用优化器对模型参数进行更新，以最小化损失函数。* optimizer.step()  
 else:  
 l.sum().backward()  
 optimizer(X.shape[0])  
 metric.add(float(l.sum()), accuracy(y\_hat, y), y.numel())  
 *# 返回训练损失和训练精度* return metric[0] / metric[2], metric[1] / metric[2]  
  
*# 加载数据*train\_data,test\_data = load\_data\_fashion\_mnist(256)  
  
for epoch in range(num\_epochs):  
 train\_loss, train\_acc = train(net, train\_data, loss, optimizer)  
 test\_acc = evaluate\_accuracy(net, test\_data)  
 print("epoch", epoch + 1, ', train\_loss', "{:.3f}".format(train\_loss), ', train\_acc', "{:.3f}".format(train\_acc), ', test\_acc', "{:.3f}".format(test\_acc))

# 输出如下：



预测的代码与之前的一致，这里就不在赘述了。

**参考文献**

《动手学深度学习》- 李沐

# 问题解答汇总

http://www.dtmao.cc/java/74093.html

https://blog.csdn.net/weixin\_44179269/article/details/124573992

<https://blog.csdn.net/qq_44665283/article/details/128336465>

<https://zhuanlan.zhihu.com/p/479135095>

[https://blog.csdn.net/h\_qiu/article/details/129203115?spm=1001.2101.3001.6650.14&utm\_medium=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7ERate-14-129203115-blog-124174526.235%5Ev27%5Epc\_relevant\_3mothn\_strategy\_and\_data\_recovery&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7ERate-14-129203115-blog-124174526.235%5Ev27%5Epc\_relevant\_3mothn\_strategy\_and\_data\_recovery&utm\_relevant\_index=18](https://blog.csdn.net/h_qiu/article/details/129203115?spm=1001.2101.3001.6650.14&utm_medium=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromBaidu~Rate-14-129203115-blog-124174526.235^v27^pc_relevant_3mothn_strategy_and_data_recovery&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromBaidu~Rate-14-129203115-blog-124174526.235^v27^pc_relevant_3mothn_strategy_and_data_recovery&utm_relevant_index=18)

[https://blog.csdn.net/qq\_42012782/article/details/121625446?spm=1001.2101.3001.6650.5&utm\_medium=distribute.pc\_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-5-121625446-blog-123478906.235%5Ev27%5Epc\_relevant\_3mothn\_strategy\_and\_data\_recovery&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-5-121625446-blog-123478906.235%5Ev27%5Epc\_relevant\_3mothn\_strategy\_and\_data\_recovery&utm\_relevant\_index=10](https://blog.csdn.net/qq_42012782/article/details/121625446?spm=1001.2101.3001.6650.5&utm_medium=distribute.pc_relevant.none-task-blog-2~default~CTRLIST~Rate-5-121625446-blog-123478906.235^v27^pc_relevant_3mothn_strategy_and_data_recovery&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~CTRLIST~Rate-5-121625446-blog-123478906.235^v27^pc_relevant_3mothn_strategy_and_data_recovery&utm_relevant_index=10)

<https://blog.csdn.net/weixin_42067873/article/details/123478906?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_baidulandingword~default-4-123478906-blog-127463720.235>^v27^pc\_relevant\_3mothn\_strategy\_and\_data\_recovery&spm=1001.2101.3001.4242.3&utm\_relevant\_index=7

[https://blog.csdn.net/qq\_45956730/article/details/127463720?spm=1001.2101.3001.6650.18&utm\_medium=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7ERate-18-127463720-blog-124174526.235%5Ev27%5Epc\_relevant\_3mothn\_strategy\_and\_data\_recovery&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7ERate-18-127463720-blog-124174526.235%5Ev27%5Epc\_relevant\_3mothn\_strategy\_and\_data\_recovery&utm\_relevant\_index=22](https://blog.csdn.net/qq_45956730/article/details/127463720?spm=1001.2101.3001.6650.18&utm_medium=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromBaidu~Rate-18-127463720-blog-124174526.235^v27^pc_relevant_3mothn_strategy_and_data_recovery&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromBaidu~Rate-18-127463720-blog-124174526.235^v27^pc_relevant_3mothn_strategy_and_data_recovery&utm_relevant_index=22)

<https://blog.csdn.net/qq_39906884/article/details/124174526>