**Softmax网络的PyTorch实现二**

在上次的文章中我们实现了Softmax网络，并使用真是的数据集进行了多分类的任务，学习过该文的小伙伴应该知道，上次的实现基本上是一步一步的来是现在整个网络的，本次文章我们继续来实现这个网络。不过不同的是，我们本次文章将基于PyTorch中的相关函数来进行实现。

1. **数据加载**

数据加载部分我们使用的与之前的一样，通过使用PyTorch中的utils.data方法来进行数据的加载：

def load\_data\_fashion\_mnist(batch\_size=None, resize=None):  
 root = "F:\DataSet\DLPytorch\Data\FashionMNIST"

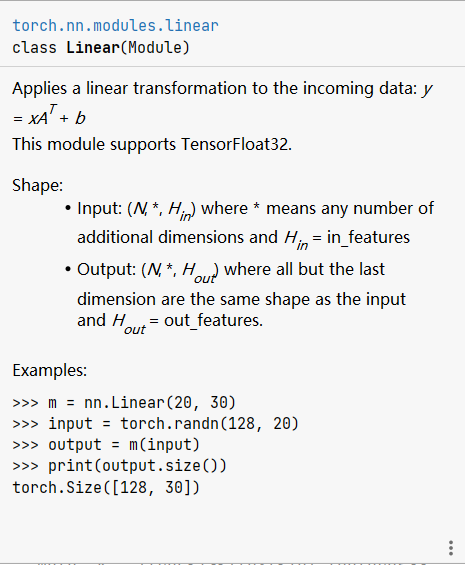
if batch\_size is None:

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))

其中，transforms.ToTensor()之前已经讲述过，其是一个归一化像素的方法，感兴趣的小伙伴自己搜索一下相关的用法。

1. **定义模型**

在本次的案例中，Softmax是一个两层的网络，且输出层是⼀个全连接层，而且我们的样本数据中也正好是一个10种类别的物品，那么我们再实现我们的模型，我们只需在Sequential中添加⼀个带有10个输出的全连接层即可，在Torch中，我们将借助torch.nn.modules.linear中的方法来进行实现，nn.Linear的用法如下所示：



上述已经说明，Softmax是一个就两层的网络，因此我们定义输入输出层即可。

in\_features = 28 \* 28  
out\_features = 10  
*# 定义模型， 全连接层*class MySoftmaxNet(nn.Module):  
 def \_\_init\_\_(self, in\_features, out\_features):  
 *# 子类继承了父类的所有属性和方法，父类属性使用父类方法来进行初始化  
 # super(MySoftmaxNet,self).\_\_init\_\_()* super().\_\_init\_\_()  
 self.linear = nn.Linear(in\_features, out\_features)

*# 向前传播* def forward(self, x):  
 *# x.shape: torch.Size([batch, 1, 28, 28])* print(x.shape)  
 return self.linear(x.view(x.shape[0], -1))  
# 定义模型  
net = MySoftmaxNet(in\_features, out\_features)

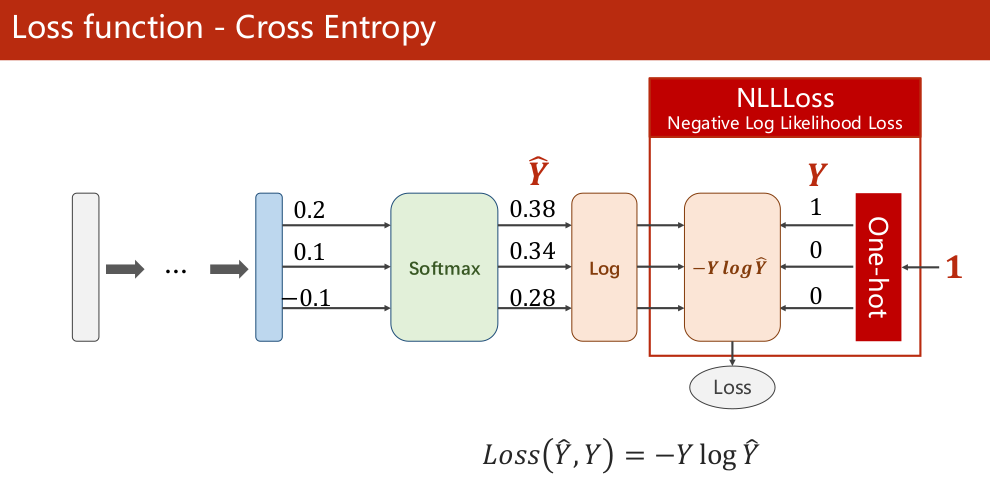
上述代码中我们设置了输入的特征数为784个，输出的种类为10，并使用前向传播。Softmax模型一开始的时候需要一定的参数，因此我们需要初始化一些我们使用的参数，这里初始化的是模型的权重和偏置参数：

*# 初始化模型参数*torch.nn.init.normal\_(net.linear.weight, mean=0, std=0.01)  
torch.nn.init.zeros\_(net.linear.bias)

这里我们使用torch.nn中的init可以快速的初始化参数，其中我们设置权重参数满足均值为0，标准差为0.01的正态分布。而偏差设置为0。

1. **损失函数定义**

损失函数这里我们选择交叉熵损失函数，其形式如下：



其中头上有标记的Y表示预测的概率分布，Y为训练集的数据。在Torch中的实现我们可以直接调用写好的代码：

loss = nn.CrossEntropyLoss(reduction='none')

这里解释一下参数reduction，该参数用于控制输出损失的形式，一般我们会遇见三种情况：

（1）当 reduction='none' 时，函数会输出一个形状为 (batch\_size, num\_classes) 的矩阵，在我们的案例中就会输出(784,10)的矩阵，其表示每个样本的每个类别的损失。

（2）当 reduction='sum' 时，函数会对矩阵求和，输出一个标量，表示所有样本的损失之和。

（3）当 reduction='elementwise\_mean' 时，函数会对矩阵求平均，输出一个标量，表示所有样本的平均损失。

1. **优化算法定义**

这里，我们使用学习率为0.1的小批量随机梯度下降作为优化算法，代码如下：

optimizer = torch.optim.SGD(net.parameters(), lr=0.1)

构建好神经网络后，网络的参数都保存在parameters()函数当中，因此我们再进行优化的时候，要使用参数就从net.parameters()中进行取。

1. **计算分类准确性**

在分类的任务重，我们把预测概率最大的类别作为输出类别，如果它与真实类别一致，说明预测正确。基于这个原理，我们进行分类准确定的计算，这一小结的描述可以参考之前文章的相关部分：

**《Softmax网络的PyTorch实现一》**

我们直接上代码：

*# 计算分类准确性*def cal\_accurary(data\_iter, net):  
 acc\_sum,n = 0.0,0  
 for X, y in data\_iter:  
 acc\_sum += (net(X).argmax(dim=1) == y).float().sum().item()  
 *# 计算准确判断的数量,shape[0]即零维度（列）的元素数量* n += y.shape[0]  
 return acc\_sum / n

1. **模型训练**

基于上述的工作，我们完成模型训练就很简单了，我们来看一下代码：

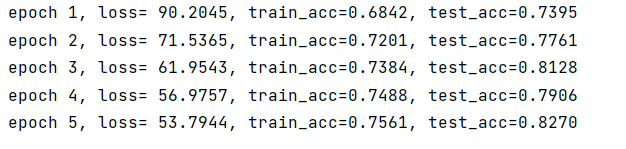
*# 定义训练函数  
# 学习周期*num\_epochs = 5  
def train(net, train\_iter, test\_iter, loss, num\_epochs, batch\_size, lr, optimizer):  
 *# 损失值、正确数量、总数* train\_loss\_sum = 0.0  
 train\_acc\_sum = 0.0  
 n = 0  
 for epoch in range(1, num\_epochs + 1):  
 for X, y in train\_iter:  
 y\_hat = net(X)  
 *# 数据集损失函数的值等于每个样本的损失函数值的和。* l = loss(y\_hat, y).sum()  
 *# 梯度清零* optimizer.zero\_grad()  
 *# 损失函数梯度计算* l.backward()  
 optimizer.step()  
  
 train\_loss\_sum += l.item()  
 train\_acc\_sum += (y\_hat.argmax(dim=1) == y).float().sum().item()  
 n += y.shape[0]  
 *# 测试集准确度* test\_acc = cal\_accurary(test\_iter, net)  
 print('epoch %d, loss= %.4f, train\_acc=%.4f, test\_acc=%.4f' % (epoch, train\_loss\_sum / n, train\_acc\_sum / n, test\_acc))

我们设置学习率和batch大小就可以进行调用了，代码如下：

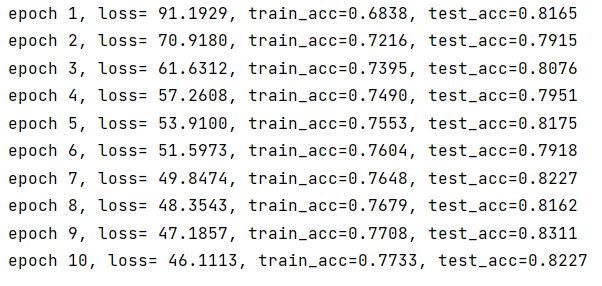
lr = 0.1  
batch\_size = 256  
train\_iter,test\_iter = load\_data\_fashion\_mnist(batch\_size = 256)  
train(net, train\_iter, test\_iter, loss, num\_epochs, batch\_size, lr, optimizer)

我们更改Epoch=5和10的时候大小的结果如下：

**Epoch 5 次的结果：**



**Epoch 10次的结果：**



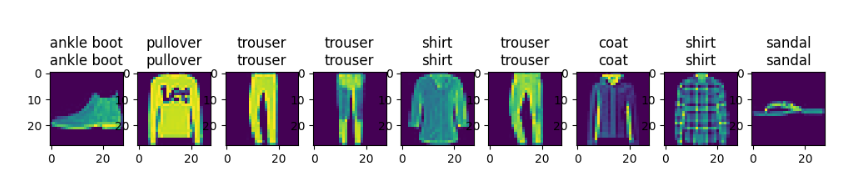
可以发现，整体上的精度在0.80附近。

1. **预测**

接下来我们进行预测，预测的同时进行图像的展示，我们的代码如下：

*# 获取数据正确的标签，即将样本的类别数字转换成文本*def get\_fmnist\_labels(labels):  
 *# 获取对应的标签* text\_labels = ['t-shirt', 'trouser', 'pullover', 'dress', 'coat', 'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']  
 return [text\_labels[int(i)] for i in labels]  
  
*# 绘制图像*from IPython import display  
def display\_images(images,labels):  
 *# 绘制矢量图b* display.display\_svg()  
 \_,figs = plt.subplots(1,len(images),figsize=(12,12))  
 *#设置添加子图的数量、大小* for f,img,label in zip(figs,images,labels):  
 f.imshow(img.view(28,28).numpy())  
  
 f.axes.get\_xaxis().set\_visible(True)  
 f.axes.get\_yaxis().set\_visible(True)  
  
 f.set\_title(label)  
 plt.show()  
  
  
*# 迭代从测试集中获得样本和标签*X, y = iter(test\_iter).next()  
print(X.shape)  
print(y.shape)  
  
*# 获取真实的标签*true\_labels = get\_fmnist\_labels(y.numpy())  
*# 获取预测的标签*pred\_labels = get\_fmnist\_labels(net(X).argmax(dim=1).numpy())  
*# 将真实标签和预测得到的标签加入到图像上*titles = [true + '\n' + pred for true, pred in zip(true\_labels, pred\_labels)]  
display\_images(X[0:9],titles[0:9])

结果如下：



至此我们就再次使用Pytorch中相关的方法实现了我们的Softmax模型。

1. **整个代码**

import torch  
import torchvision  
import torch.nn as nn  
from matplotlib import pyplot as plt  
from torch.utils import data  
from torchvision import transforms  
  
*# 数据加载*def load\_data\_fashion\_mnist(batch\_size=None, resize=None):  
  
 root = "E:\DataSet\DLPytorch\Data\FashionMNIST"  
  
 if batch\_size is None:  
 batch\_size = 256  
 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))  
  
  
*# 输入输出*in\_features = 28 \* 28  
out\_features = 10  
  
*# 定义模型， 全连接层*class MySoftmaxNet(nn.Module):  
 def \_\_init\_\_(self, in\_features, out\_features):  
 *# 子类继承了父类的所有属性和方法，父类属性使用父类方法来进行初始化  
 # super(MySoftmaxNet,self).\_\_init\_\_()* super().\_\_init\_\_()  
 self.linear = nn.Linear(in\_features, out\_features)  
  
 *# 向前传播* def forward(self, x):  
 *# x.shape: torch.Size([batch, 1, 28, 28])  
 # print(x.shape)* return self.linear(x.view(x.shape[0], -1))  
  
*# 定义模型*net = MySoftmaxNet(in\_features, out\_features)  
*# 初始化模型参数*torch.nn.init.normal\_(net.linear.weight, mean=0, std=0.01)  
torch.nn.init.zeros\_(net.linear.bias)  
  
*# 损失函数定义*loss = nn.CrossEntropyLoss(reduction='none')  
  
*# 优化器*optimizer = torch.optim.SGD(net.parameters(), lr=0.1)  
  
*# 计算分类准确性*def cal\_accurary(data\_iter, net):  
 acc\_sum,n = 0.0,0  
 for X, y in data\_iter:  
 acc\_sum += (net(X).argmax(dim=1) == y).float().sum().item()  
 *# 计算准确判断的数量,shape[0]即零维度（列）的元素数量* n += y.shape[0]  
 return acc\_sum / n  
  
  
*# 定义训练函数  
# 学习周期*num\_epochs = 5  
*# num\_epochs = 5*def train(net, train\_iter, test\_iter, loss, num\_epochs, batch\_size, lr, optimizer):  
 *# 损失值、正确数量、总数* train\_loss\_sum = 0.0  
 train\_acc\_sum = 0.0  
 n = 0  
 for epoch in range(1, num\_epochs + 1):  
 for X, y in train\_iter:  
 y\_hat = net(X)  
 *# 数据集损失函数的值等于每个样本的损失函数值的和。* l = loss(y\_hat, y).sum()  
 *# 梯度清零* optimizer.zero\_grad()  
 *# 损失函数梯度计算* l.backward()  
 optimizer.step()  
  
 train\_loss\_sum += l.item()  
 train\_acc\_sum += (y\_hat.argmax(dim=1) == y).float().sum().item()  
 n += y.shape[0]  
 *# 测试集准确度* test\_acc = cal\_accurary(test\_iter, net)  
 print('epoch %d, loss= %.4f, train\_acc=%.4f, test\_acc=%.4f' % (epoch, train\_loss\_sum / n, train\_acc\_sum / n, test\_acc))  
  
  
lr = 0.1  
batch\_size = 256  
  
train\_iter,test\_iter = load\_data\_fashion\_mnist(batch\_size = 256)  
train(net, train\_iter, test\_iter, loss, num\_epochs, batch\_size, lr, optimizer)  
  
  
*# 获取数据正确的标签，即将样本的类别数字转换成文本*def get\_fmnist\_labels(labels):  
 *# 获取对应的标签* text\_labels = ['t-shirt', 'trouser', 'pullover', 'dress', 'coat', 'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']  
 return [text\_labels[int(i)] for i in labels]  
  
*# 绘制图像*from IPython import display  
def display\_images(images,labels):  
 *# 绘制矢量图b* display.display\_svg()  
 \_,figs = plt.subplots(1,len(images),figsize=(12,12))  
 *#设置添加子图的数量、大小* for f,img,label in zip(figs,images,labels):  
 f.imshow(img.view(28,28).numpy())  
  
 f.axes.get\_xaxis().set\_visible(True)  
 f.axes.get\_yaxis().set\_visible(True)  
  
 f.set\_title(label)  
 plt.show()  
  
  
*# 迭代从测试集中获得样本和标签*X, y = iter(test\_iter).next()  
print(X.shape)  
print(y.shape)  
  
*# 获取真实的标签*true\_labels = get\_fmnist\_labels(y.numpy())  
*# 获取预测的标签*pred\_labels = get\_fmnist\_labels(net(X).argmax(dim=1).numpy())  
*# 将真实标签和预测得到的标签加入到图像上*titles = [true + '\n' + pred for true, pred in zip(true\_labels, pred\_labels)]  
  
display\_images(X[0:9],titles[0:9])

1. **其他**

动手学深度学习中,使用的是Sequence来实现模型的，即：

in\_features = 28 \* 28  
out\_features = 10

*# 定义模型， 全连接层*class MySoftmaxNet(nn.Module):  
 def \_\_init\_\_(self, in\_features, out\_features):  
 *# 子类继承了父类的所有属性和方法，父类属性使用父类方法来进行初始化  
 # super(MySoftmaxNet,self).\_\_init\_\_()* super().\_\_init\_\_()  
 self.linear = nn.Linear(in\_features, out\_features)  
  
 *# 向前传播* def forward(self, x):  
 *# x.shape: torch.Size([batch, 1, 28, 28])  
 # print(x.shape)* return self.linear(x.view(x.shape[0], -1))  
  
*# 定义模型*net = MySoftmaxNet(in\_features, out\_features)  
*# 初始化模型参数*torch.nn.init.normal\_(net.linear.weight, mean=0, std=0.01)  
torch.nn.init.zeros\_(net.linear.bias)

**可以更改为以下代码的实现：**

in\_features = 28 \* 28  
out\_features = 10  
  
*# net = nn.Sequential(nn.Flatten(), nn.Linear(784, 10))*net = nn.Sequential(nn.Flatten(), nn.Linear(in\_features, out\_features))  
def init\_weights(m):  
 if type(m) == nn.Linear:  
 nn.init.normal\_(m.weight, std=0.01)  
 nn.init.zeros\_(m.bias)  
*# 调用*net.apply(init\_weights);

这两种实现是一致的，在简洁的角度上来看还是第二种比较好。

**十、参考文献**

（1）《动手学深度学习》

https://zhuanlan.zhihu.com/p/197623117

<https://zhuanlan.zhihu.com/p/210037407>

<https://zhuanlan.zhihu.com/p/108344765>

<http://www.ay1.cc/article/2191.html>

<https://blog.csdn.net/qq_40806950/article/details/126453269>

<https://blog.csdn.net/wltsysterm/article/details/104440387>

<https://blog.csdn.net/weixin_36331429/article/details/112701199>

https://blog.51cto.com/u\_15127616/4297856