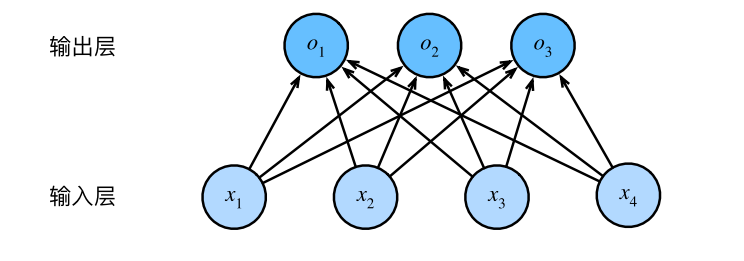
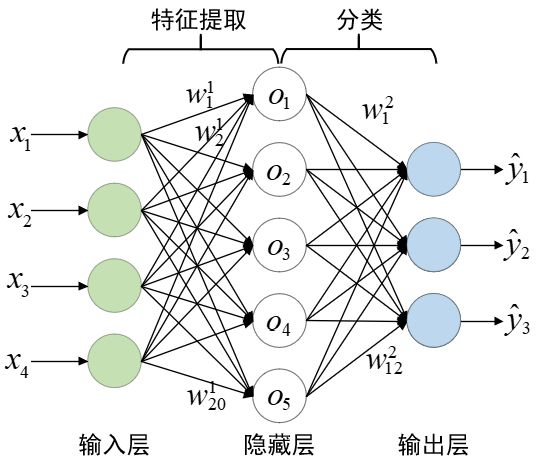
**Softmax网络的PyTorch实现一**

**一、Softmax的基本结构**

在实际的问题中，我们的模型需要多个输出（比如多分类，手写数字的识别），每一个类别对应一个输出。

我们今天的主角softmax网络的结构如下，其实一个单层的神经网络，输出层由输入层来进行决定，从网络结构可以看出其是一个全连接层的神经网络。这里不过多剖析网络的结构，更多的相关知识大家可以参考一些机器学习的教程。





接下来我们来实现这个网络。

**二、Softmax的实现**

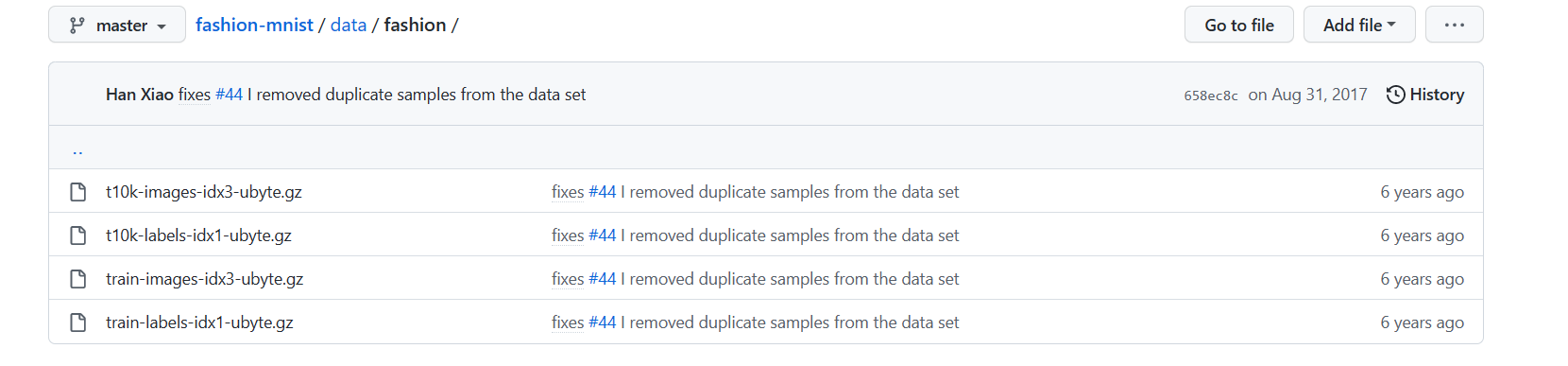
我们的大致流程如下：

**2.1 数据集Fashion-MNIST准备**

我们将使⽤Fashion-MNIST数据集来进行实验，一些小伙伴可能知道MNIST数据集。MNIST数据集中的数据都是一些手写数字的图像，而Fashion-MNIST中的数据集是我们常见一些服饰类的物品。包含的10个类别：t-shirt（T恤）、trouser（裤⼦）、pullover（套衫）、dress（连⾐裙）、coat（外套）、sandal（凉鞋）、shirt（衬衫）、sneaker（运动鞋）、bag（包）和ankle boot（短靴），注意Fashion-MNIST数据集中的图片是灰度图。

我们可以通过这个链接去下载该数据集：

<https://github.com/zalandoresearch/fashion-mnist>>



也可以使用torchvision.datasets中的方法来下载这些数据图片，代码如下，其中root路径为保存的数据的路径：

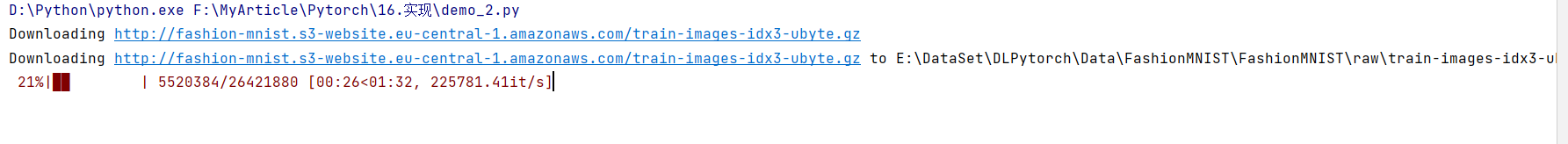
import torchvision  
from torchvision import transforms  
  
def load\_dataset():  
 fmnist\_train = torchvision.datasets.FashionMNIST(root="E:\DataSet\DLPytorch\Data\FashionMNIST", train=True, download=True,transform=transforms.ToTensor())  
 fmnist\_test = torchvision.datasets.FashionMNIST(root="E:\DataSet\DLPytorch\Data\FashionMNIST", train=False, download=True,transform=transforms.ToTensor())  
 return fmnist\_train, fmnist\_test  
  
fmnist\_train, fmnist\_test = load\_dataset()  
print(len(fmnist\_train))  
print(len(fmnist\_test))

# 输出

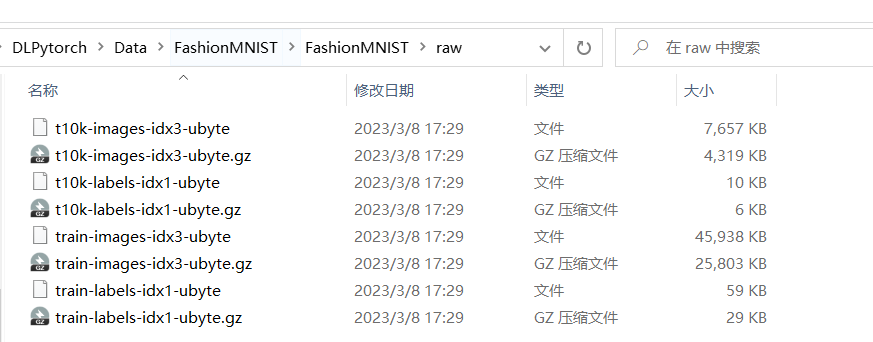
60000

10000

下载过程如下所示：



下载之后我们可以在路径下查看到数据：



从打印的训练集和测试集的长度可以看出，训练集和测试集共有60000个样本和10000个样本。

我们可以进一步查看每一张图片的形状：

print(type(fmnist\_train))  
print(len(fmnist\_train[0]))  
print(fmnist\_train[0][0].shape)  
print(fmnist\_train[0][1])

# 输出：

<class 'torchvision.datasets.mnist.FashionMNIST'>

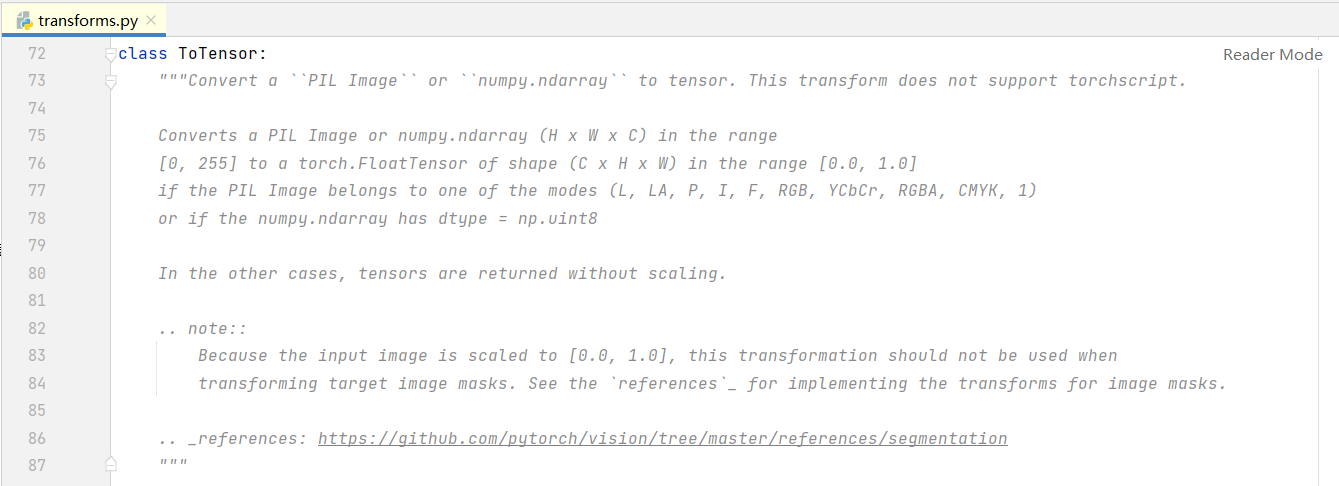
2

torch.Size([1, 28, 28])

9

可以看出，该样本数据的通道数为1（即灰度图），宽度和高度均为28个像素。且该样本数据的标签为9，对应到一个具体的类别。这里主要注意一下download=True这个参数，表示数据集不存在目标目录的时候进行下载。这里稍微注意fmnist\_train[0][0]表示测试集的第一个元素（第一张图片），fmnist\_train[0][1]表示第一张图片所属的标签（标签是数字形式的）。

关于transforms.ToTensor()的作用这里跟大家分享一下：



概括一下就是ToTensor的作用是将导入的图片转换为Tensor的格式，导入的图片为PIL image 或者 numpy.nadrry格式的图片，其shape为（H-W-C）数值范围在[0,255]，转换之后shape为（C-H-W）,数值范围被规约在[0,1]之间（即将每一个像素值都除以255），例如：

import numpy as np  
from torchvision import transforms  
  
a = np.random.random((28,28,3))  
transform = transforms.Compose([transforms.ToTensor()])  
b = transform(a)  
print(a.shape)  
print(b.shape)

# 输出

(28, 28, 3)

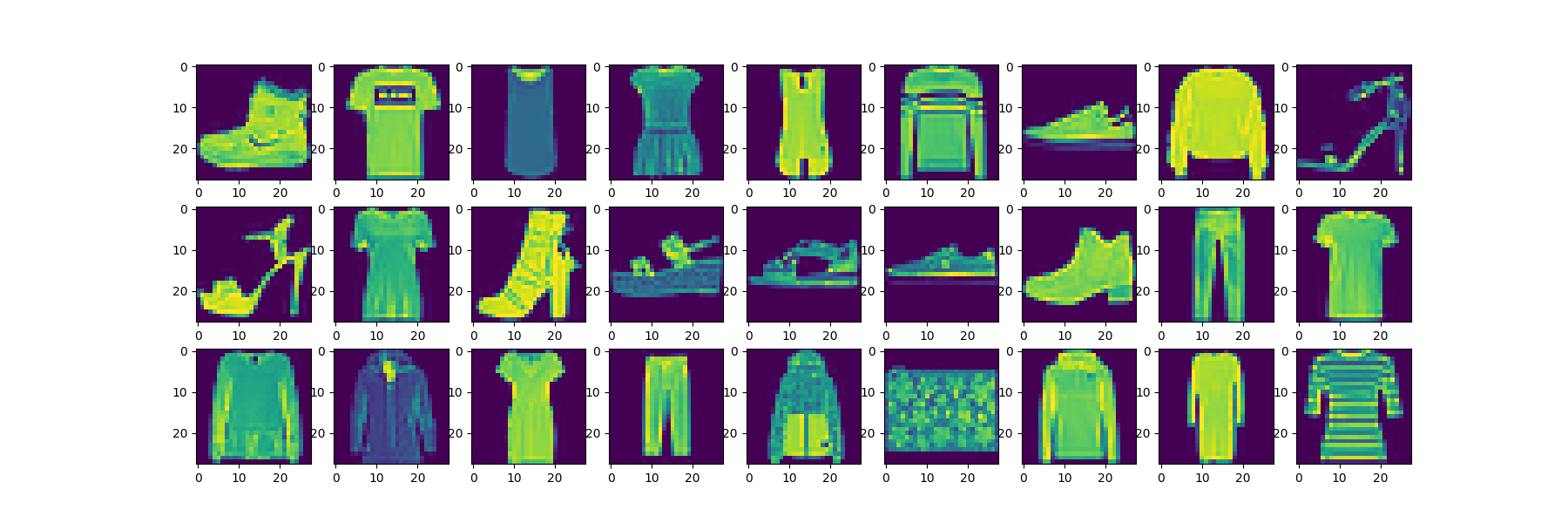
torch.Size([3, 28, 28])

**2.2 数据集Fashion-MNIST的可视化**

接下来我们试着对其中的样本进行可视化展示，代码如下：

def display\_images(imgs, num\_rows, num\_cols, scale=2.0):  
 *# 设置图像画图大小* figsize = (num\_cols \* scale, num\_rows \* scale)  
 \_, axes = plt.subplots(num\_rows, num\_cols, figsize=figsize)  
 axes = axes.flatten()  
 for i, (ax, img) in enumerate(zip(axes, imgs)):  
 if torch.is\_tensor(img):  
 *# 图⽚张量* ax.imshow(img.numpy())  
 else:  
 *# PIL图⽚* ax.imshow(img)  
 ax.axes.get\_xaxis().set\_visible(True)  
 ax.axes.get\_yaxis().set\_visible(True)  
 *# 显示图片* plt.show()  
 return axes  
  
*# 这里加载的是整个测试集fmnist\_train，所以设置bacth\_size获取，并使用iter进行获取*X, y = next(iter(data.DataLoader(fmnist\_train, batch\_size=3\*9)))  
axes = display\_images(X.reshape(3\*9, 28, 28), 3, 9)

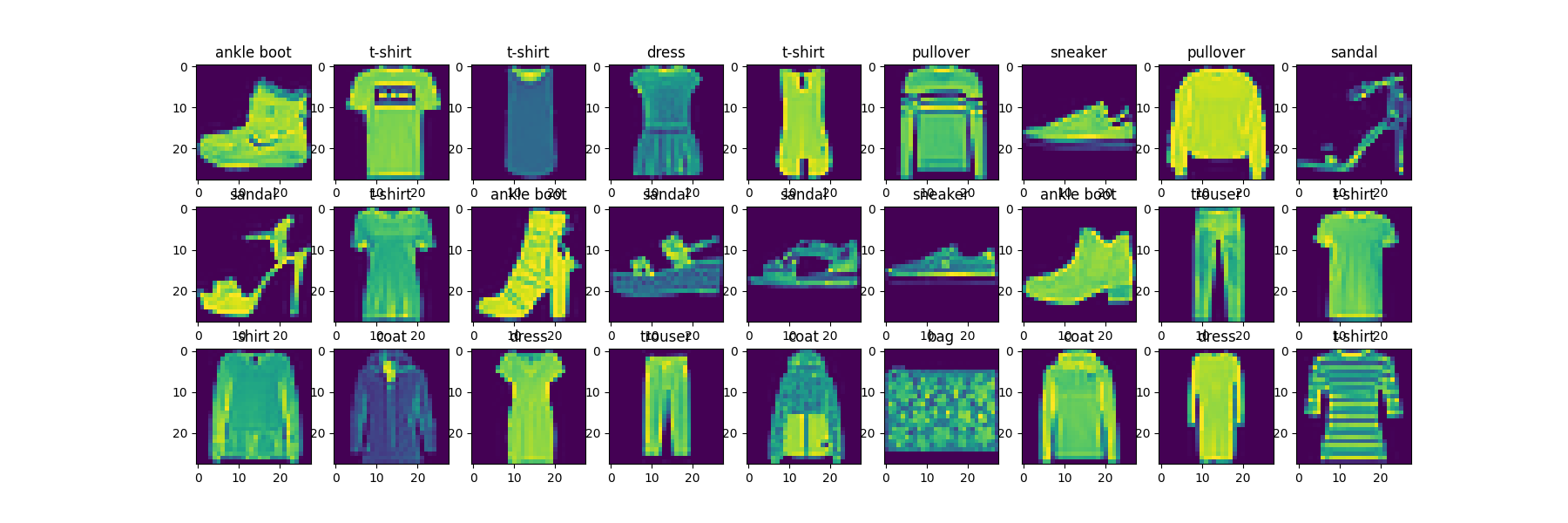
# 结果如下



batch\_size=3\*9，正好是3行9列数据，因为这个测试数据集的真实标签我们是知道的，因此可以将真实的标签加入到上述图片中，具体的代码如下：

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]  
  
  
def display\_images(imgs, num\_rows, num\_cols, scale=2.0,titles=None,):  
 *# 设置图像画图大小* figsize = (num\_cols \* scale, num\_rows \* scale)  
 \_, axes = plt.subplots(num\_rows, num\_cols, figsize=figsize)  
 axes = axes.flatten()  
 for i, (ax, img) in enumerate(zip(axes, imgs)):  
 if torch.is\_tensor(img):  
  *# 图⽚张量* ax.imshow(img.numpy())  
 else:  
 *# PIL图⽚* ax.imshow(img)  
 ax.axes.get\_xaxis().set\_visible(True)  
 ax.axes.get\_yaxis().set\_visible(True)  
 if titles:  
 ax.set\_title(titles[i])  
 *# 显示图片* plt.show()  
 return axes  
  
*# 这里加载的是整个测试集fmnist\_train，所以设置bacth\_size获取，并使用iter进行获取*X, y = next(iter(data.DataLoader(fmnist\_train, batch\_size=3\*9)))  
axes = display\_images(X.reshape(3\*9, 28, 28), 3, 9, titles=get\_fmnist\_labels(y))

# 图片展示如下：



其中，axes = axes.flatten()将axes由n\*m的Axes组展平成1\*nm的axes组,形成一张整体图片。

**2.3 批量加载数据集**

上述我们使用的是iter一次获取27张图片进行展示的，实际工程上我们需要遍历整个train和test数据集的，在每次迭代中，数据加载器每次都会读取⼀⼩批量数据，⼤⼩为batch\_size，

这样在后续的训练中我们可以进行批处理。因此我们使用dataLoader方式加载图片，这里直接参考《动手学深度学习》中的代码片段：

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))  
  
train\_iter, test\_iter = load\_data\_fashion\_mnist(32, resize=64)  
print(len(train\_iter))  
print(len(test\_iter))  
for X, y in train\_iter:  
 print(X.shape, X.dtype, y.shape, y.dtype)  
 break

# 输出为：

1875

313

torch.Size([32, 1, 64, 64]) torch.float32 torch.Size([32]) torch.int64

上述代码主要作用是批量加载我们的训练数据和测试数据，并设置每一批次的数据为32张图片，且将每一张图片的大小resize为64，原始的数据的大小为28\*28。train\_iter变量中存储着所有的训练集合数据共1875\*32 = 60000张，test\_iter变量中存储着所有的测试数据共312\*32 + 16 =9984 + 16=10000，16张图片单独成1个batch。至此，我们就准备好了数据集。

**2.4 模型参数初始化**

对于图像数据的输入，我们一般是将整个图片打平进行输入，即将28\*28的图片，打平为长度为784的一维向量。因为数据集共有10和类别，因此我们设置Softmax回归方法的权重W的维度为784\*10，偏置项B为1\*10的行向量，代码如下所示：

num\_inputs = 784  
num\_outputs = 10  
  
W = torch.normal(0, 0.01, size=(num\_inputs, num\_outputs), requires\_grad=True)  
b = torch.zeros(num\_outputs, requires\_grad=True)  
print(len(W))  
print(len(b))

**# 输出**

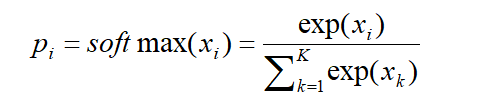
**784**

**10**

**2.5 Softmax函数实现**

接下来我们一起看看Softmax函数。Softmax将分类预测输出进行标准化，其将某一个样本点计算归属于类别i的概率为：





该函数表达式的代码实现如下所示：

def softmax(X):  
 X\_exp = torch.exp(X)  
 partition = X\_exp.sum(1, keepdim=True)  
 return X\_exp / partition

这里解释一下第3行代码，X.sum(dim=1, keepdim=True)表示对X中的元素按照行进行求和，比如：

X = torch.tensor([[1,2,3], [4,5,6],[7,8,9]])  
print(X.sum(dim=0, keepdim=True))   
print(X.sum(dim=1, keepdim=True))

**# 输出为**

tensor([[12, 15, 18]])

tensor([[ 6],

[15],

[24]])

因为预测结果输出是概率，因此对于每一个样本其输出标签的值的总和（即概率）为1，比如我们使用上述定义的softmax代码：

def softmax(X):  
 X\_exp = torch.exp(X)  
 partition = X\_exp.sum(1, keepdim=True)  
 return X\_exp / partition  
  
X = torch.tensor([[1,2,3], [4,5,6],[7,8,9]])  
res = softmax(X)  
print(res)  
print(res.sum(1))

**# 输出**

tensor([[0.0900, 0.2447, 0.6652],

[0.0900, 0.2447, 0.6652],

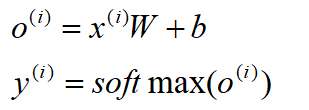
[0.0900, 0.2447, 0.6652]])

tensor([1., 1., 1.])

这样我们就定义好了Softmax方法，接下来我们实现Softmax模型。

**2.6 Softmax回归模型**

Softmax模型的可以描述为：





即将输出使用softmax进行映射即可，代码实现如下所示：

def softmaxnet(X):  
 return softmax(torch.mm(X.view((-1, num\_inputs)), W) + b)

这个使用了torch.mm()方法，即我们在大学的高等代数学习的矩阵乘法。torch.mul()是矩阵元素的对应相乘。该方法可以参考官方文档：

<https://pytorch.org/docs/stable/generated/torch.mm.html>

X.view((-1, num\_inputs)表示将X打成固定的形状，-1表示不确定行数，确定的num\_inputs列。这个Softmax模型看起来还是很清晰的。

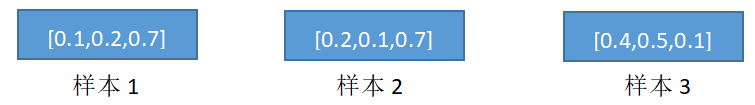
**2.7 损失函数定义**

既然我们要构建一个模型，那损失函数一定是少不了的。这里我们使用常见的交叉熵损失函数，该损失函数也是深度学习中最常见的损失函数。大家可以参考机器学习书籍来学习相关的损失函数原型。先给出多分类的交叉损失函数的实现代码：

def cross\_entropy(y\_hat, y):  
 return - torch.log(y\_hat[range(len(y\_hat)), y])

为什么可以这样来实现呢？我们举一个例子：

假设现在我们的模型对三个样本进行预测，且预测得到的概率分布分别为:



三个样本的正确的标签分布为[1,0,0]、[0,1,0]和[0,0,1]，那么根据交叉熵的计算公式，我们此时的交叉熵为：-(1\*log0.1 + 1\*log0.1 + 1\*log0.1)。其余的为0，这里就没有乘上去。

对于测试集的样本而言，我们是知道其正确的标签的，因此在这种情况下，只需要通过正确的标签映射到具体的预测概率分布值，再取对数进加法操作即可。比如：

y = torch.tensor([0, 1, 2])  
y\_hat = torch.tensor([  
 [0.1, 0.2, 0.7],  
 [0.2, 0.1, 0.7],  
 [0.4, 0.5, 0.1]  
])  
print(y\_hat[[0, 1, 2], y])

# 输出

tensor([0.1000, 0.1000, 0.1000])

这里的y表示样本类别预测概率对应的真实标签，上述脚本的意思就是从y\_hat的第一个样本中取第0个元素，从第二个样本中取第1个元素，从第三个样本中取第2个元素，对应的都是0.1, 0.1和0.1，**代码y\_hat[[0, 1, 2], y]是使用了索引的方式进行取值的。至此我们就完成了损失函数的实现。**

**2.8 准确度定义**

定义完了损失函数，接下来就是定义准确度了。我们知道对于输出的概率的问题，对于每个样本的预测类别，我们会选择概率值最大的类别作为预测的结果进行输出，代码如下：

def accuracy(y\_hat,y):  
 acc = (y\_hat.argmax(1) == y).float().mean()  
 return acc.item()

我们来使用一下这个函数：

y = torch.tensor([0, 1, 2])  
y\_hat = torch.tensor([  
 [0.1, 0.2, 0.7],  
 [0.2, 0.1, 0.7],  
 [0.4, 0.5, 0.1]  
])  
print(accuracy(y\_hat,y))

# 输出为

0.0

对代码和输出进行解释：

1. y\_hat.argmax(1) ：表示取预测值中最大值对应的标签。
2. y\_hat.argmax(1) == y：表示预测的标签与真实的标签比对。
3. float：将比对的bool值转换为0.0或1.0的浮点型数据。
4. mean()：用于求解均值。

上述案例中，由于真实的标签是[0,1,2]，y\_hat中3个输出概率分布中最大值依次为0.7,0.7和0.5，其对应的标签依次为2,2和1。由于0≠2,且1≠2且2≠1，因此结果输出为0。

《动手学深度学习》中，关于accuracy的实现如下，表达的意思是一样的，这里贴一下代码：

def accuracy(y\_hat, y):if len(y\_hat.shape) > 1 and y\_hat.shape[1] > 1:  
 y\_hat = y\_hat.argmax(axis=1)  
 cmp = y\_hat.type(y.dtype) == y  
 return float(cmp.type(y.dtype).sum())

**2.9 模型评估**

在训练集上训练好模型之后，我们需要在测试集上进行数据的评估，看看模型到底能预测多少准确，此时就要求在测试集上计算其总的准确率：

def evaluate\_accuracy(test\_data\_iter, softmaxnet):   
 acc\_sum, n = 0.0, 0  
 for X, y in test\_data\_iter:

# 预测  
 y\_hat = softmaxnet(X)  
 acc\_sum += (y\_hat.argmax(dim=1) == y).float().sum().item()  
 n += y.shape[0]  
 return acc\_sum / n

上述代码很好理解，就是对批量的数据集使用模型之后再计算总体的准确度。

**2.10 模型训练**

最后一步就是模型的训练了，我们需要在训练集上进行学习（训练），首先写一个梯度下降的优化函数：

def mysgd(params, lr, batch\_size):  
 with torch.no\_grad():  
 for param in params:  
 param -= lr \* param.grad / batch\_size  
 param.grad.zero\_()

接着我们写训练函数：

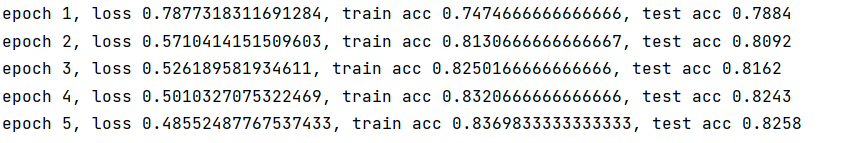
def my\_softmax\_train(softmaxnet, train\_iter, test\_iter, cross\_entropy, num\_epochs, batch\_size,  
 params=None, lr=None, optimizer=None):  
 *# 循环数* for epoch in range(num\_epochs):  
 train\_l\_sum, train\_acc\_sum, n = 0.0, 0.0, 0  
 for X, y in train\_iter:  
 y\_hat = softmaxnet(X)  
 l = cross\_entropy(y\_hat, y).sum()  
 *# 梯度清零* if optimizer is not None:  
 optimizer.zero\_grad()  
 elif params is not None and params[0].grad is not None:  
 for param in params:  
 param.grad.data.zero\_()  
 *# 反向传播优化* l.backward()  
 if optimizer is None:  
 mysgd(params, lr, batch\_size)  
 else:  
 optimizer.step()  
 *# 统计准确率* train\_l\_sum += l.item()  
 train\_acc\_sum += (y\_hat.argmax(dim=1) == y).sum().item()  
 n += y.shape[0]  
 test\_acc = evaluate\_accuracy(test\_iter, softmaxnet)  
 print(f"epoch {epoch + 1}, loss {(train\_l\_sum / n)}, train acc {train\_acc\_sum / n}, test acc {test\_acc}")

相关代码就不解释了，比较清晰，我们直接来进行调用：

batch\_size = 256  
num\_epochs = 5  
lr = 0.1

train\_iter, test\_iter = load\_data\_fashion\_mnist(256)  
my\_softmax\_train(softmaxnet, train\_iter, test\_iter, cross\_entropy, num\_epochs, batch\_size, [W, b], lr)

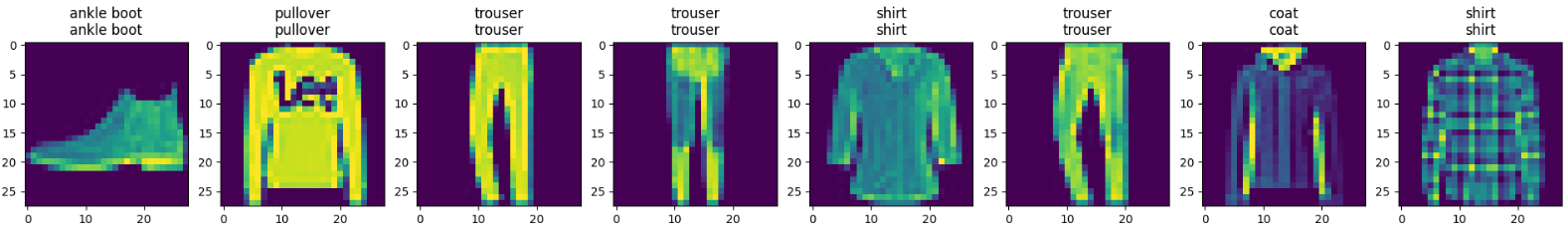
# 输出如下：



可以看出在测试集的上预测准确率达到了82%以上。我们最后来进行预测：

*# 取一次测试集的数据*def predict\_test(softmaxnet, test\_iter, n=9):  
 for X, y in test\_iter:  
 break  
 trues = get\_fmnist\_labels(y)  
 preds = get\_fmnist\_labels(softmaxnet(X).argmax(axis=1))  
 titles = [true +'\n' + pred for true, pred in zip(trues, preds)]  
 display\_images(X[0:n].reshape((n, 28, 28)), 1, n, titles=titles[0:n])  
  
predict\_test(softmaxnet,test\_iter)

# 输出如下：



从图片中可以看出，预测与标签基本上是一致的，上述几张图片是正确的。至此我们就实现了softmax网络来进行预测。

**三、总结**

上述我们实现了softmax的实现，基本上是一步一步开始实现的，贴一下所有的代码：

import torch  
import torchvision  
from matplotlib import pyplot as plt  
from torch.utils import data  
from torchvision import transforms  
  
  
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]  
  
  
def display\_images(imgs, num\_rows, num\_cols, scale=3,titles=None,):  
 *# 设置图像画图大小* figsize = (num\_cols \* scale, num\_rows \* scale)  
 \_, axes = plt.subplots(num\_rows, num\_cols, figsize=figsize)  
 axes = axes.flatten()  
 for i, (ax, img) in enumerate(zip(axes, imgs)):  
 if torch.is\_tensor(img):  
 *# 图⽚张量* ax.imshow(img.numpy())  
 else:  
 *# PIL图⽚* ax.imshow(img)  
 ax.axes.get\_xaxis().set\_visible(True)  
 ax.axes.get\_yaxis().set\_visible(True)  
 if titles:  
 ax.set\_title(titles[i])  
 *# 显示图片* plt.show()  
 return axes  
  
  
def load\_data\_fashion\_mnist(batch\_size, resize=None):  
 root = "F:\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))  
  
num\_inputs = 784  
num\_outputs = 10  
W = torch.normal(0, 0.01, size=(num\_inputs, num\_outputs), requires\_grad=True)  
b = torch.zeros(num\_outputs, requires\_grad=True)  
  
def softmax(X):  
 X\_exp = torch.exp(X)  
 partition = X\_exp.sum(1, keepdim=True)  
 return X\_exp / partition  
  
def softmaxnet(X):  
 return softmax(torch.mm(X.view((-1, num\_inputs)), W) + b)  
 *# return softmax(torch.matmul(X.reshape((-1, W.shape[0])), W) + b)*def cross\_entropy(y\_hat, y):  
 return - torch.log(y\_hat[range(len(y\_hat)), y])  
  
def evaluate\_accuracy(test\_data\_iter, softmaxnet):  
 acc\_sum, n = 0.0, 0  
 for X, y in test\_data\_iter:  
 *# 预测* y\_hat = softmaxnet(X)  
 acc\_sum += (y\_hat.argmax(dim=1) == y).float().sum().item()  
 n += y.shape[0]  
 return acc\_sum / n  
  
def mysgd(params, lr, batch\_size):  
 with torch.no\_grad():  
 for param in params:  
 param -= lr \* param.grad / batch\_size  
 param.grad.zero\_()  
  
def my\_softmax\_train(softmaxnet, train\_iter, test\_iter, cross\_entropy, num\_epochs, batch\_size,  
 params=None, lr=None, optimizer=None):  
 *# 循环数* for epoch in range(num\_epochs):  
 train\_l\_sum, train\_acc\_sum, n = 0.0, 0.0, 0  
 for X, y in train\_iter:  
 y\_hat = softmaxnet(X)  
 l = cross\_entropy(y\_hat, y).sum()  
 *# 梯度清零* if optimizer is not None:  
 optimizer.zero\_grad()  
 elif params is not None and params[0].grad is not None:  
 for param in params:  
 param.grad.data.zero\_()  
 *# 反向传播优化* l.backward()  
 if optimizer is None:  
 mysgd(params, lr, batch\_size)  
 else:  
 optimizer.step()  
 *# 统计准确率* train\_l\_sum += l.item()  
 train\_acc\_sum += (y\_hat.argmax(dim=1) == y).sum().item()  
 n += y.shape[0]  
 test\_acc = evaluate\_accuracy(test\_iter, softmaxnet)  
 print(f"epoch {epoch + 1}, loss {(train\_l\_sum / n)}, train acc {train\_acc\_sum / n}, test acc {test\_acc}")  
  
*# 参数设置*batch\_size = 256  
num\_epochs = 5  
lr = 0.1  
  
*# 加载数据*train\_iter, test\_iter = load\_data\_fashion\_mnist(256)  
*# 模型训练*my\_softmax\_train(softmaxnet, train\_iter, test\_iter, cross\_entropy, num\_epochs, batch\_size, [W, b], lr)  
  
*# 取一次测试集的数据，进行测试*def predict\_test(softmaxnet, test\_iter, n=9):  
 for X, y in test\_iter:  
 break  
 trues = get\_fmnist\_labels(y)  
 preds = get\_fmnist\_labels(softmaxnet(X).argmax(axis=1))  
 titles = [true +'\n' + pred for true, pred in zip(trues, preds)]  
 display\_images(X[0:n].reshape((n, 28, 28)), 1, n, titles=titles[0:n])  
  
predict\_test(softmaxnet,test\_iter)

1. **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