```python
# -*- coding: utf-8 -*-
# @Time    : 2024/3/8
# @Author  : quanchenliu
# @Function: 从零实现 Softmax

import torch
from d2l import torch as d2l
from IPython import display
import matplotlib.pyplot as plt

"""2、定义Softmax操作"""
def softmax(X):
    X_exp = torch.exp(X)                            # 对每一项求幂，[]
    partition = X_exp.sum(1, keepdim=True)          #
    return X_exp / partition                        #


"""3、定义模型"""
def net(X):
    # 先将 X 展开为一个 256 * 784 的向量，batch_size = 256
    prediction = torch.matmul(X.reshape(-1, W.shape[0]), W) + b     # 计算未规范化的预测
    return softmax(prediction)                                      # 使用 softmax 进行规范化处理


"""4、定义损失函数"""
def cross_entropy(y_hat, y):
    # range(len(y_hat)) 生成一个从 0 到 len(y_hat)-1 的整数列表，表示样本的索引
    # y 是真实标签，其中每个元素表示相应样本的真实类别索引，其形状为：[256, ]
    # 通过使用两个整数列表作为索引，可以同时选择张量的多个位置的元素：
    # y_hat[range(len(y_hat)), y]: 选取 y_hat 张量中第一个列表中的元素作为行索引，第二个列表中的元素作为列索引，返回对应位置的元素
    return -torch.log(y_hat[range(len(y_hat)), y])


"""5、求正确预测数"""
def accuracy(y_hat, y):          # @save
    if len(y_hat.shape) > 1 and y_hat.shape[1] > 1:          # 如果 y_hat 是矩阵
        y_hat = y_hat.argmax(axis=1)                         # 取 1 维/第二个维度（每一行）中的最大元素的索引
    cmp = y_hat.type(y.dtype) == y                           # 将预测类别与真实类别进行比较，结果是一个只有0、1的张量
    return float(cmp.type(y.dtype).sum())                    # 降维求和，得到预测准确的数量
```

```python
"""6、评估在任意模型上的精度"""
def evaluate_accuracy(net, data_iter):
    if isinstance(net, torch.nn.Module):
        net.eval()                                          # 将模型设为评估
模式
    metric = Accumulator(2)
    with torch.no_grad():
        # 使用 net(X) 计算预测值，并调用 accuracy 函数计算预测正确的样本数
        # y.numel() 用于计算预测总数
        # 放入 Accumulator 中进行叠加，使得在遍历数据集的时候，二者随着时间的推
移而增加
        for X, y in data_iter:
            metric.add(accuracy(net(X), y), y.numel())
    return metric[0] / metric[1]

class Accumulator:      # @save
    def __init__(self, n):
        self.data = [0.0] * n

    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]


"""7、绘制图表的类：用于可视化训练进度"""
class Animator:     # @save
    def __init__(self, xlabel=None, ylabel=None, legend=None, xlim=None,
ylim=None,
                 xscale='linear', yscale='linear', fmts=('-', 'm--', 'g-.',
'r:'),
                 nrows=1, ncols=1, figsize=(3.5, 2.5)):
        if legend is None:
            legend = []
        d2l.use_svg_display()
        self.fig, self.axes = d2l.plt.subplots(nrows, ncols,
figsize=figsize)
        if nrows * ncols == 1:
            self.axes = [self.axes, ]
        self.config_axes = lambda: d2l.set_axes(
            self.axes[0], xlabel, ylabel, xlim, ylim, xscale, yscale,
legend)
        self.X, self.Y, self.fmts = None, None, fmts

    def add(self, x, y):
        if not hasattr(y, "__len__"):
            y = [y]
        n = len(y)
```

```
        if not hasattr(x, "__len__"):
            x = [x] * n
        if not self.X:
            self.X = [[] for _ in range(n)]
        if not self.Y:
            self.Y = [[] for _ in range(n)]

        for i, (a, b) in enumerate(zip(x, y)):
            if a is not None and b is not None:
                self.X[i].append(a)
                self.Y[i].append(b)
        self.axes[0].cla()
        for x, y, fmt in zip(self.X, self.Y, self.fmts):
            self.axes[0].plot(x, y, fmt)
        self.config_axes()
        display.display(self.fig)
        display.clear_output(wait=True)


"""8、训练"""
def train_epoch_ch3(net, train_iter, loss, updater):        # @save
    if isinstance(net, torch.nn.Module):
        net.train()

    metric = Accumulator(3)
    for X, y in train_iter:
        y_hat = net(X)
        l = loss(y_hat, y)
        if isinstance(updater, torch.optim.Optimizer):
            updater.zero_grad()
            l.mean().backward()
            updater.step()
        else:
            l.sum().backward()
            updater(X.shape[0])
        metric.add(float(l.sum()), accuracy(y_hat, y), y.numel())
    return metric[0] / metric[2], metric[1] / metric[2]

def train_ch3(net, train_iter, test_iter, loss, num_epochs, updater):   #
@save
    global train_metric, test_acc
    animator = Animator(xlabel='epoch', xlim=[1, num_epochs], ylim=[0.3,
0.9],
                        legend=['train loss', 'train acc', 'test acc'])
    for epoch in range(num_epochs):
        train_metric = train_epoch_ch3(net, train_iter, loss, updater)
        test_acc = evaluate_accuracy(net, test_iter)
        animator.add(epoch+1, train_metric + (test_acc, ))
        if epoch == 9:
            plt.show()          # 显示绘制的最后一张图表
```

```
    train_loss, train_acc = train_metric
    assert train_loss < 0.5, train_loss
    assert 1 >= train_acc > 0.7, train_acc
    assert 1 >= test_acc > 0.7, train_acc


"""10、预测"""
def predict_ch3(net, test_iter, n=6): # @save
    for X, y in test_iter:
        break
    trues = d2l.get_fashion_mnist_labels(y)
    preds = d2l.get_fashion_mnist_labels(net(X).argmax(axis=1))
    titles = [true + '\n' + pred for true, pred in zip(trues, preds)]
    d2l.show_images(X[0:n].reshape((n, 28, 28)), 1, n, titles=titles[0:n])


# 用于更新模型参数的函数
def updater(batch_size):
    return d2l.sgd([w, b], lr, batch_size)

def main():
    # 1、初始化模型参数
    global w, b, lr, train_metric, test_acc
    batch_size = 256            # 批处理大小为 256
    num_inputs = 784            # 28 * 28 =784, 即: 将28*28的图像展平，得到
一个长度为784的向量
    num_outputs = 10            # 分类结果有10个类，因此输出维度为10
    w = torch.normal(0, 0.01, size=(num_inputs, num_outputs),
requires_grad=True)   # [784, 10]
    b = torch.zeros(num_outputs, requires_grad=True)
         # [1, 10]
    train_iter, test_iter = d2l.load_data_fashion_mnist(batch_size)
        # 加载数据时，每个批次的数据包含 batch_size 个样本

    # 9、训练模型
    lr = 0.1                    # 学习率设为0.1
    num_epochs = 10             # 训练轮次设为10
    train_ch3(net, train_iter, test_iter, cross_entropy, num_epochs,
updater)

    # 11、预测
    predict_ch3(net, test_iter)


if __name__ == "__main__":
    main()
```