**暨南大学本科实验报告专用纸**

课程名称 深度学习实验 成绩评定

实验项目名称 Softmax回归与K折验证 指导教师 林聪

实验项目编号 01 实验项目类型 实验地点 C304

学生姓名 赵俊文 学号 2022104002

学院 智能科学与工程 系 专业 人工智能

实验时间 2025 年 4 月 4 日 午～ 4 月 13 日 午 温度 ℃湿度

1. **实验目的**
2. 理解并实现Softmax回归模型
3. 掌握K折交叉验证的原理与实现方法
4. 比较Softmax层在不同位置对模型性能的影响
5. 在FashionMNIST数据集上实现K折交叉验证训练流程
6. **主要仪器设备**

**仪器：** PC

**实验环境：**

windows

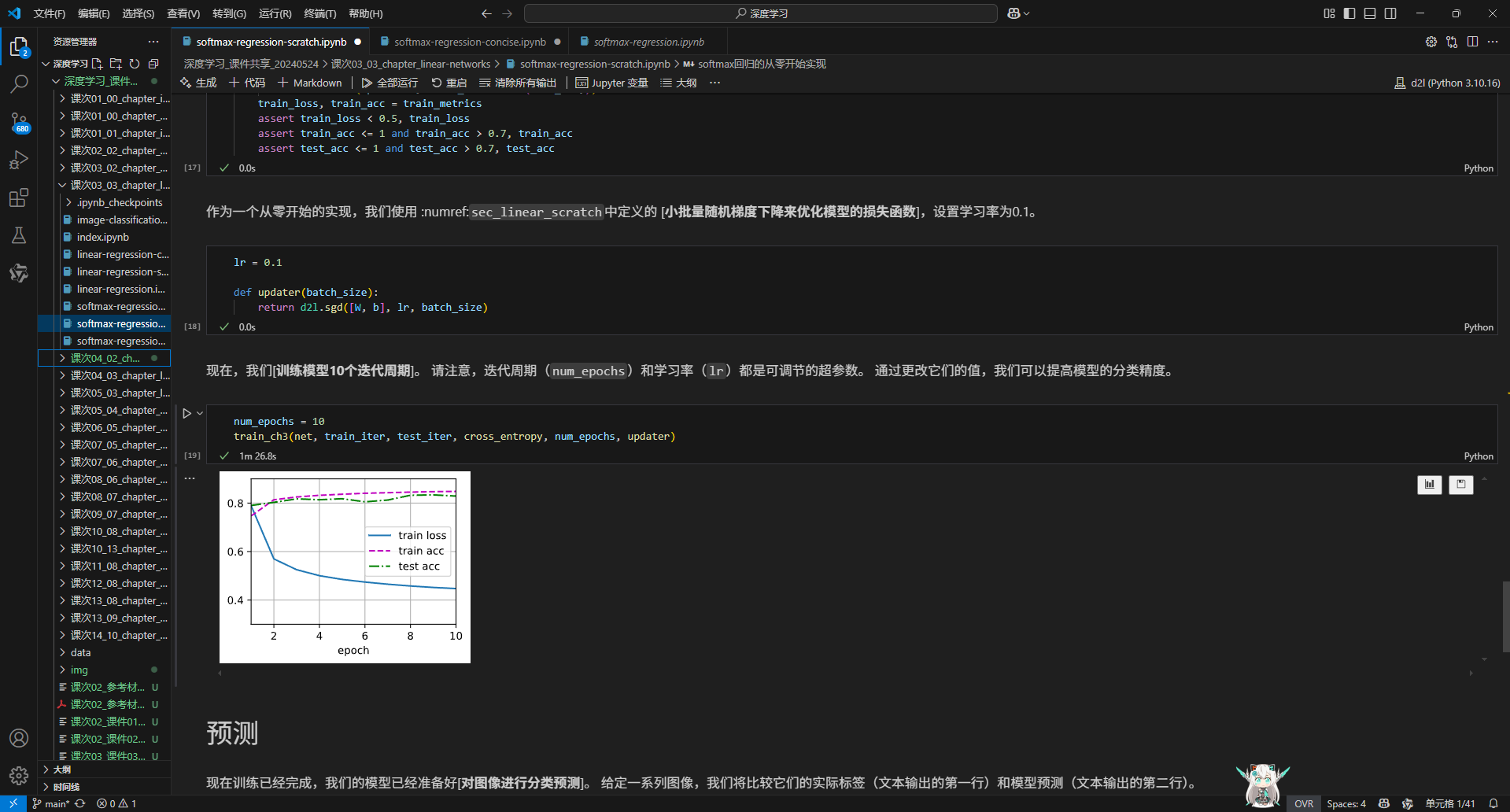
Python 3.10.6

PyTorch 2.7.0

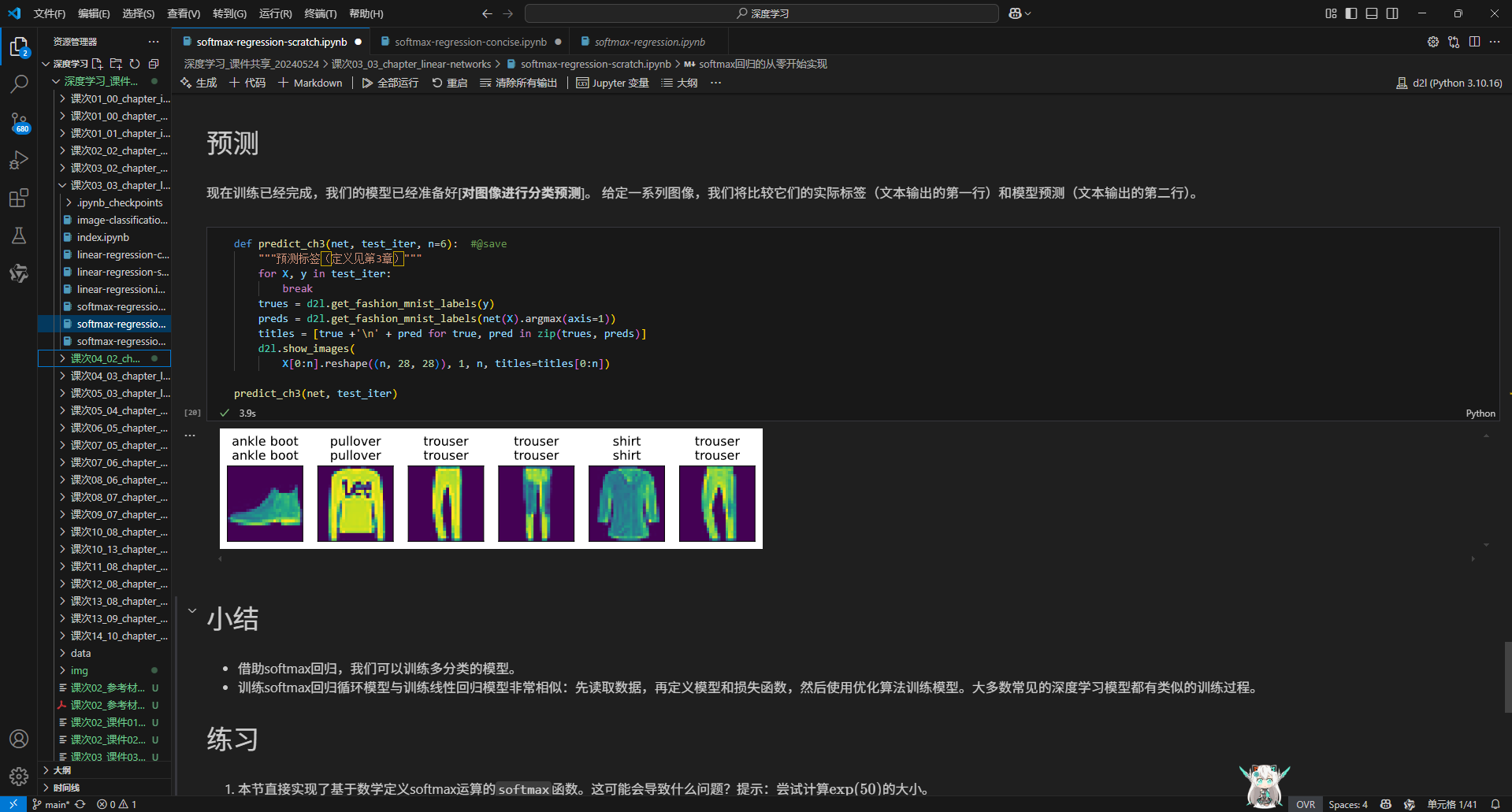
1. **实验步骤与调试**
2. **运行相关章节的代码**
3. softmax回归的从零开始实现

所有代码运行截图过于繁琐，因此只保留一些必要的运行结果之图片

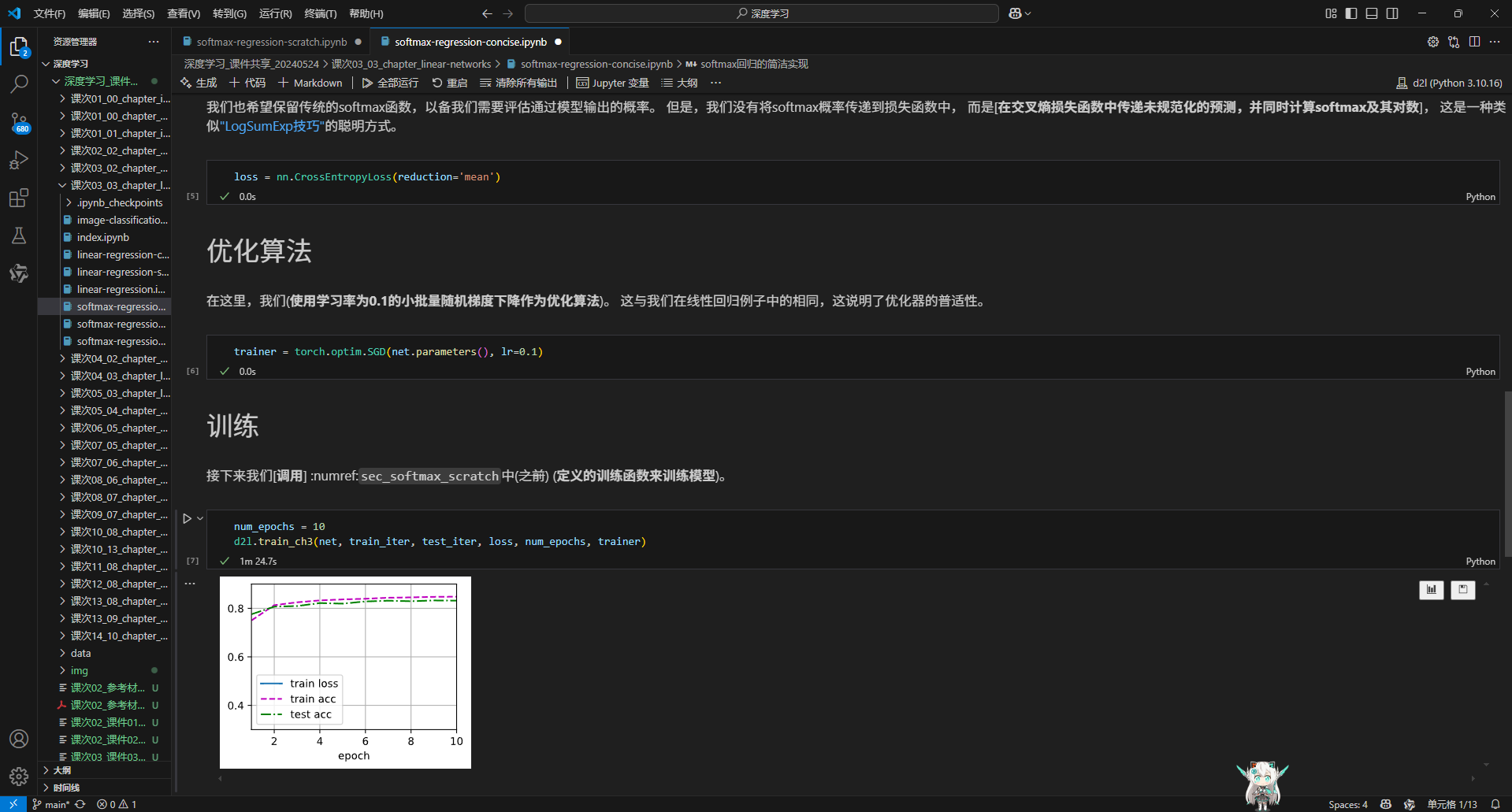
1. 训练



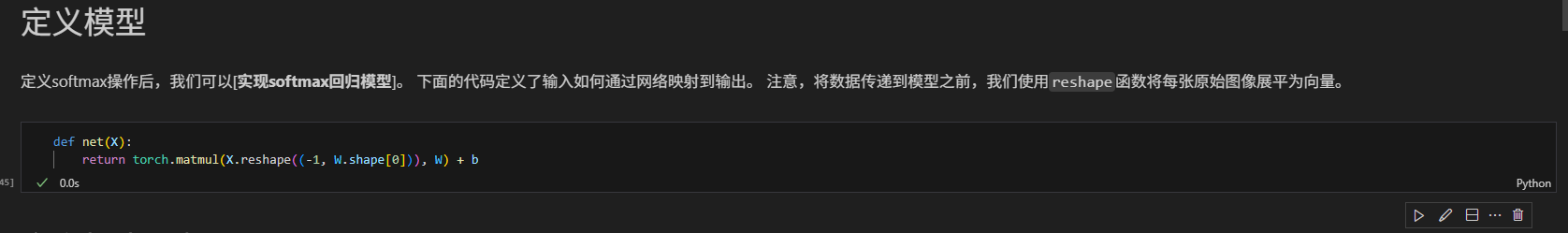
1. 预测



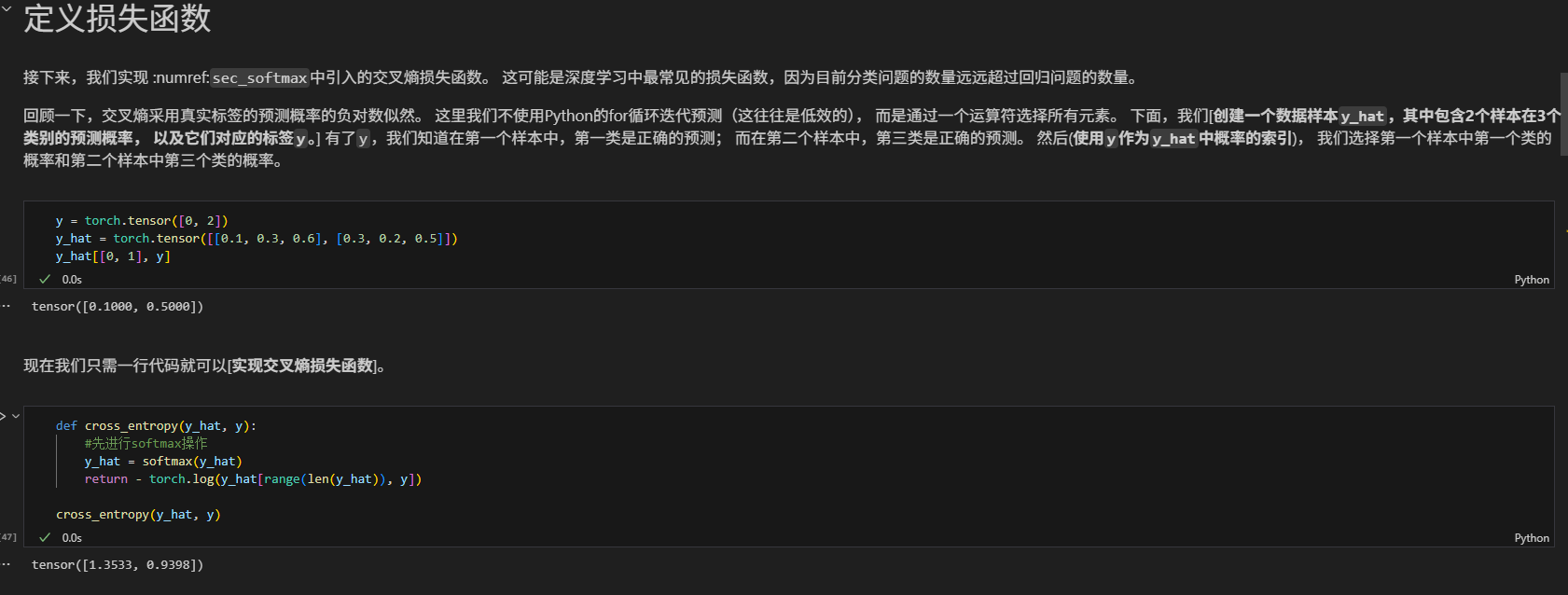
1. softmax回归的简洁实现



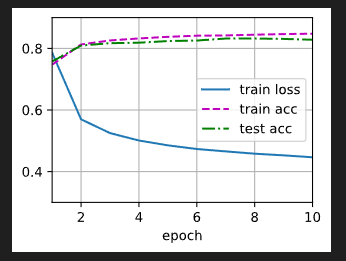
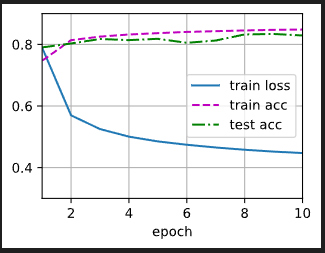
1. **调整Sofmax层的位置,到损失函数中，对比准确性性能**
2. 最后一次输出不再使用sofmax,改为直接输出前面层的计算结果



1. 损失函数先进行softmax，然后再进行交叉熵损失计算

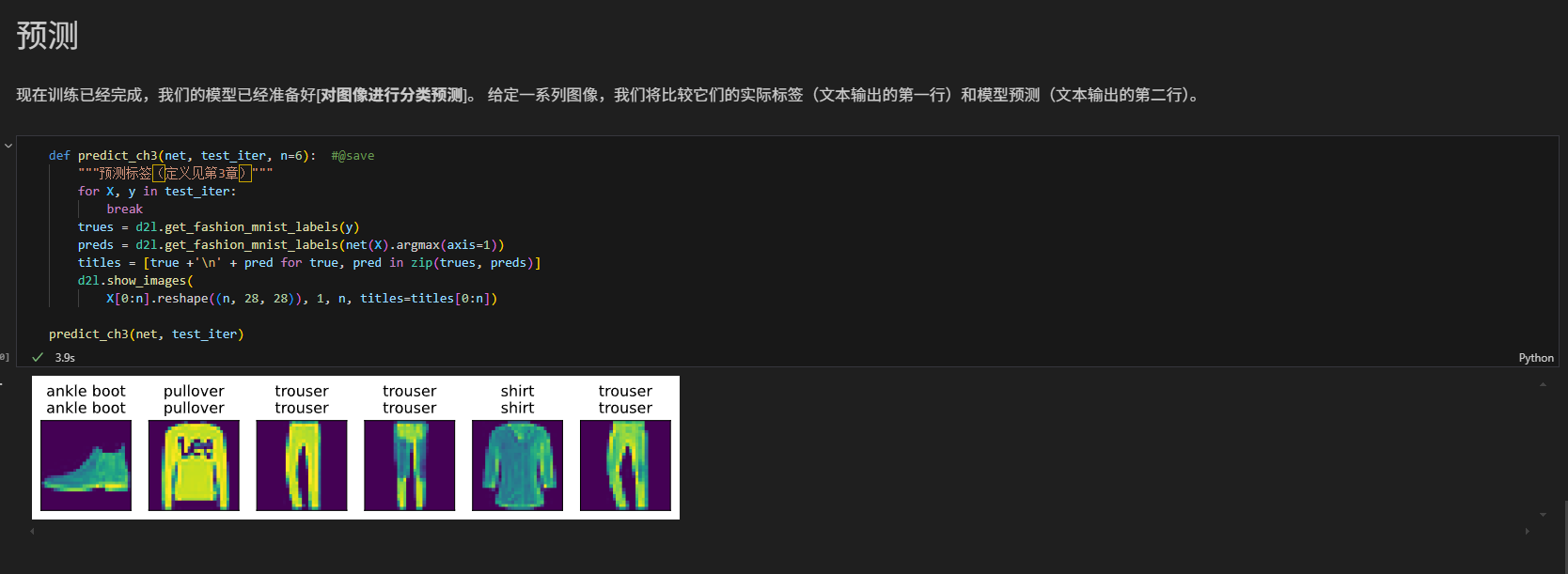


1. 运行结果对比
2. 训练结果对比（左为修改前）

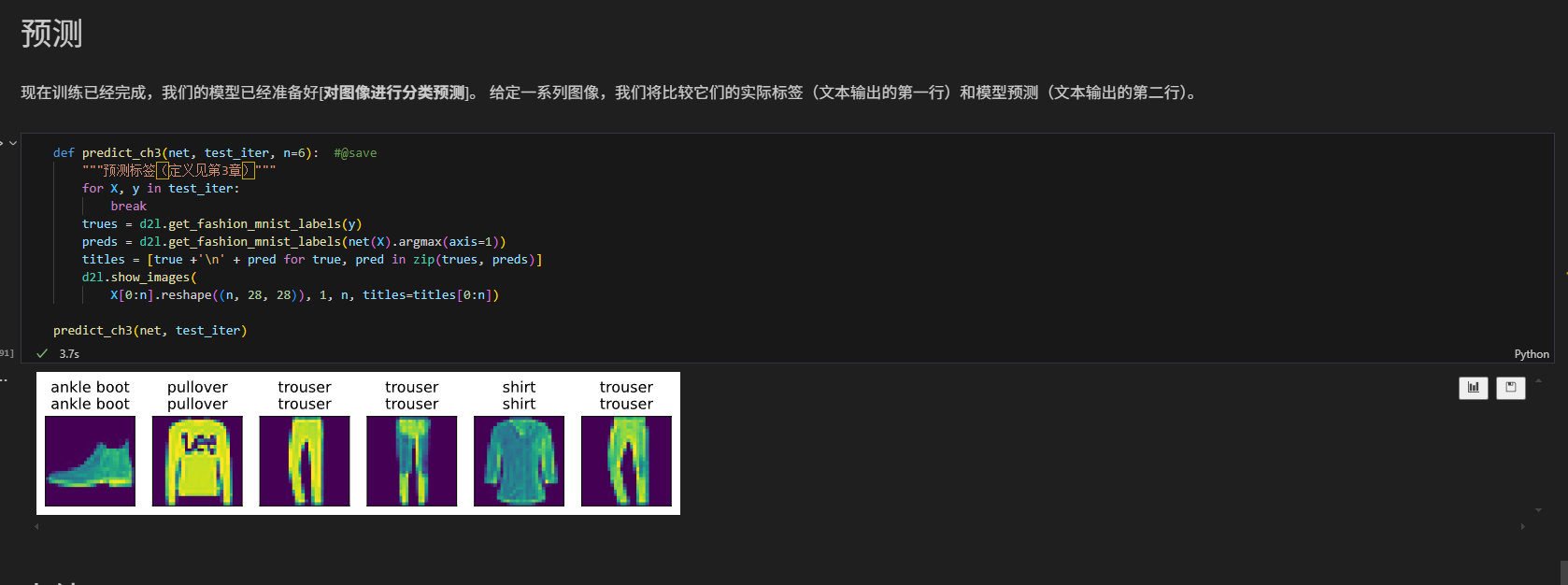


1. 预测结果对比

修改前



修改后



1. **K折交验证**

* 采用gpu版本的Pytorch框架进行计算验证，数据集采用fashionMNIST数据集
* 模型采用Pytorch提供的简单神经网络模型
* K=5

1. 源代码

import torch

import torchvision

from torch.utils.data import DataLoader, Subset

from torchvision import transforms

from sklearn.model\_selection import KFold

from d2l import torch as d2l

# 数据预处理

trans = transforms.ToTensor()

mnist\_train = torchvision.datasets.FashionMNIST(

    root="../data", train=True, transform=trans, download=True)

mnist\_test = torchvision.datasets.FashionMNIST(

    root="../data", train=False, transform=trans, download=True)

# 定义简单神经网络模型

class SimpleModel(torch.nn.Module):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.flatten = torch.nn.Flatten()

        self.linear\_relu\_stack = torch.nn.Sequential(

            torch.nn.Linear(28\*28, 512),

            torch.nn.ReLU(),

            torch.nn.Linear(512, 512),

            torch.nn.ReLU(),

            torch.nn.Linear(512, 10),

        )

    def forward(self, x):

        x = self.flatten(x)

        logits = self.linear\_relu\_stack(x)

        return logits

# K折交叉验证设置

k = 5

kf = KFold(n\_splits=k, shuffle=True, random\_state=42)

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# 训练和验证循环

def train\_and\_validate(model, train\_loader, val\_loader, num\_epochs=10):

    criterion = torch.nn.CrossEntropyLoss()

    optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

    for epoch in range(num\_epochs):

        model.train()

        for X, y in train\_loader:

            X, y = X.to(device), y.to(device)

            optimizer.zero\_grad()

            outputs = model(X)

            loss = criterion(outputs, y)

            loss.backward()

            optimizer.step()

    model.eval()

    correct, total = 0, 0

    with torch.no\_grad():

        for X, y in val\_loader:

            X, y = X.to(device), y.to(device)

            outputs = model(X)

            \_, predicted = torch.max(outputs.data, 1)

            total += y.size(0)

            correct += (predicted == y).sum().item()

    return correct / total

fold\_accuracies = []

for fold, (train\_idx, val\_idx) in enumerate(kf.split(mnist\_train)):

    print(f"\nFold {fold + 1}")

    # 创建数据加载器

    train\_subset = Subset(mnist\_train, train\_idx)

    val\_subset = Subset(mnist\_train, val\_idx)

    train\_loader = DataLoader(train\_subset, batch\_size=256, shuffle=True, num\_workers=4)

    val\_loader = DataLoader(val\_subset, batch\_size=256, shuffle=False, num\_workers=4)

    # 初始化模型

    model = SimpleModel().to(device)

    # 训练并验证

    accuracy = train\_and\_validate(model, train\_loader, val\_loader)

    fold\_accuracies.append(accuracy)

    print(f"Fold {fold+1} Validation Accuracy: {accuracy\*100:.2f}%")

# 输出结果

print("\nK-Fold Cross Validation Results:")

for fold, acc in enumerate(fold\_accuracies):

    print(f"Fold {fold+1}: {acc\*100:.2f}%")

print(f"Average Accuracy: {sum(fold\_accuracies)/k\*100:.2f}%")

1. 分类结果（正确率）

