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

（一） 实验目的

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

（二） 主要仪器设备

仪器： PC

实验环境：

windows

Python 3.10.6

PyTorch 2.7.0

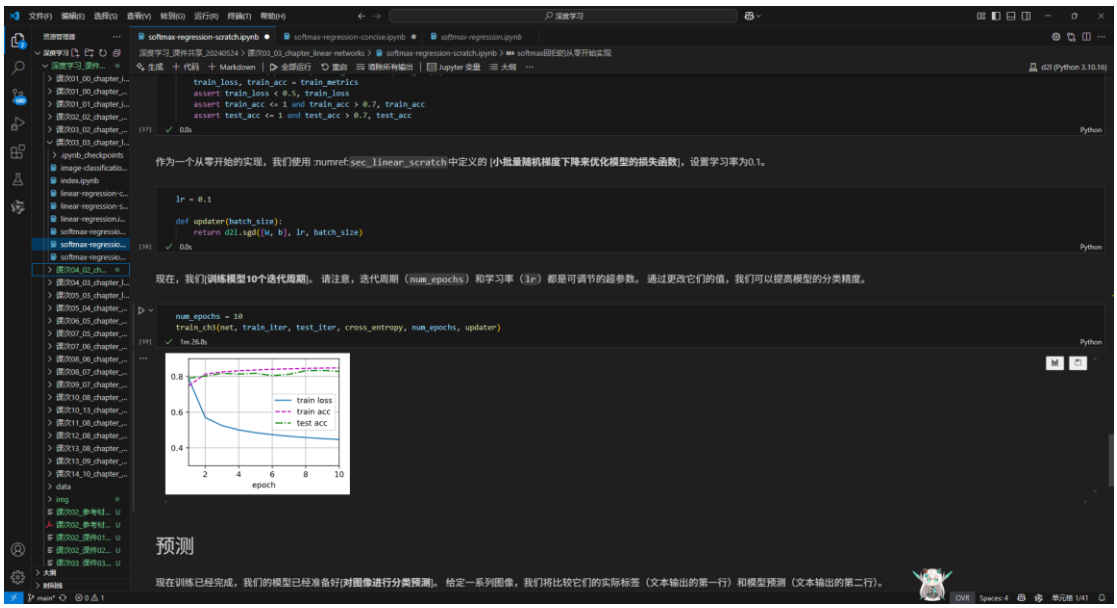
（三） 实验步骤与调试

1. 运行相关章节的代码

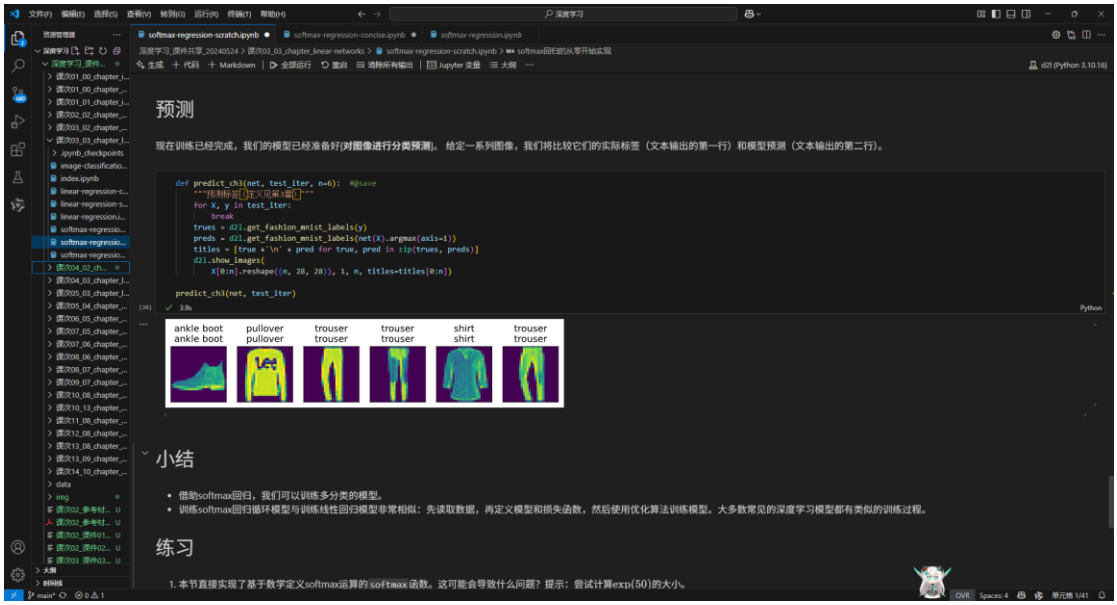
(1) softmax 回归的从零开始实现

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

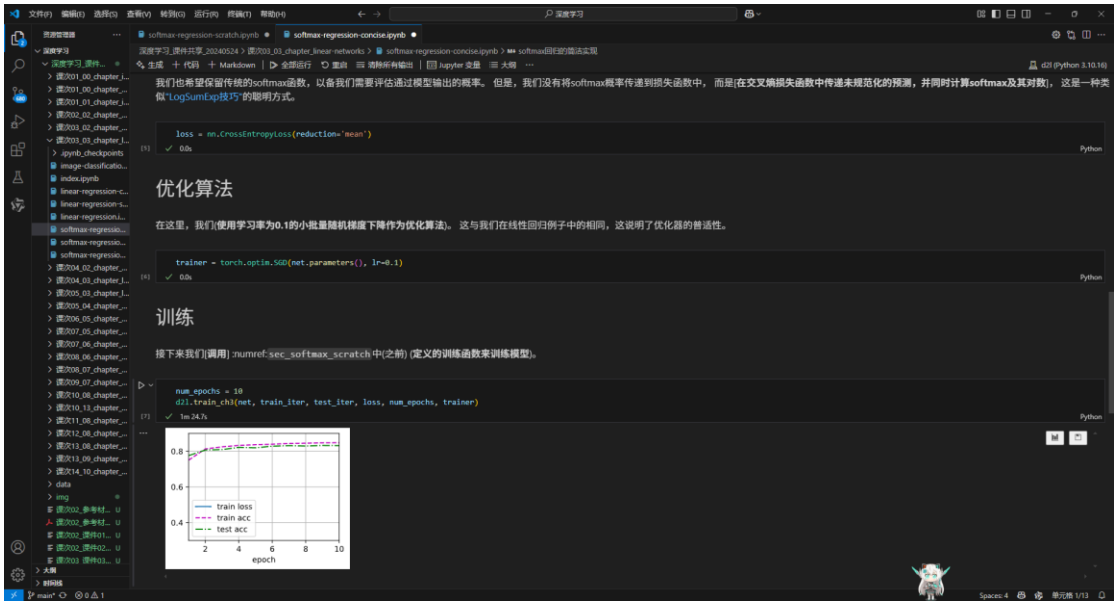
a) 训练



b) 预测



(2) softmax 回归的简洁实现



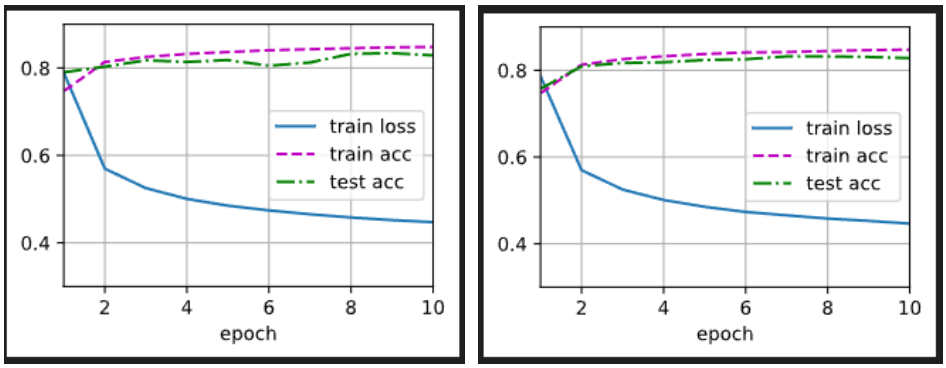
2. 调整 Sofmax 层的位置,到损失函数中，对比准确性性能

(1) 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

(2) 损失函数先进行 softmax，然后再进行交叉熵损失计算

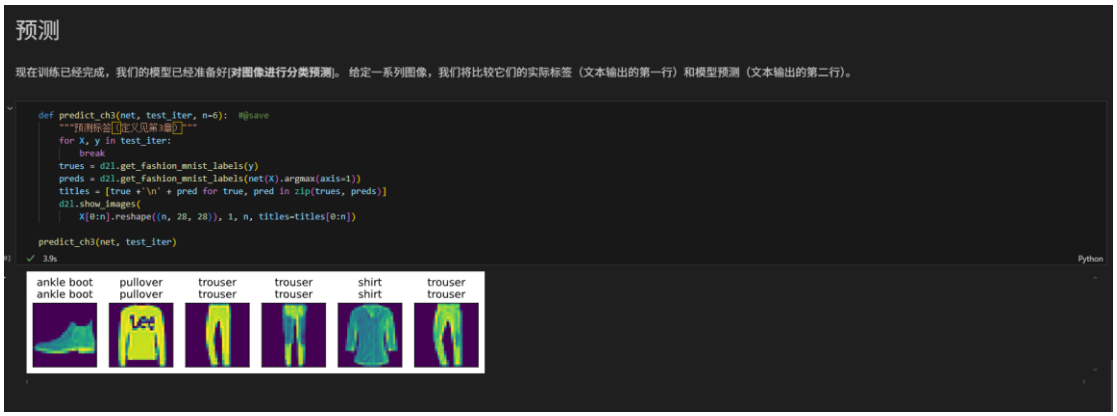
(3) 运行结果对比

a) 训练结果对比（左为修改前）

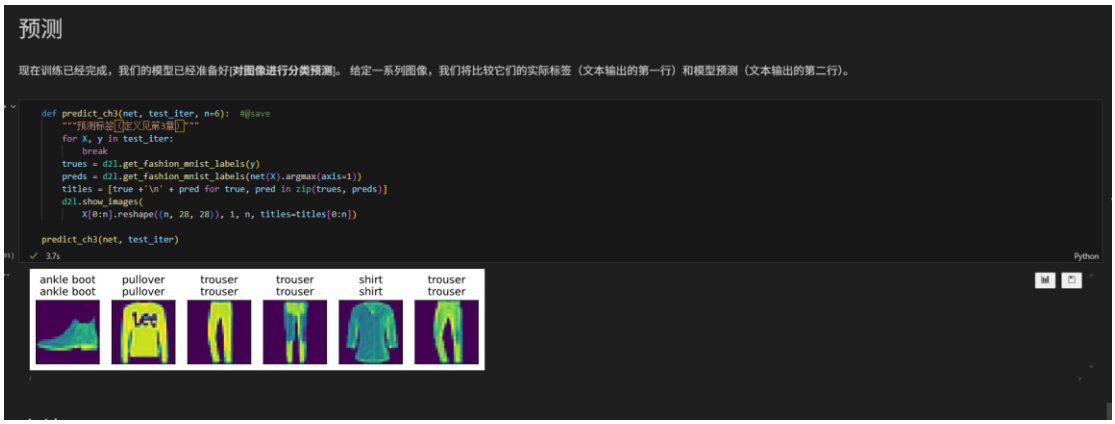


b) 预测结果对比

修改前



修改后



3. 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}%")
```

(2) 分类结果（正确率）