**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 验证型 实验地点

学生姓名 王秀喆 学号 2022100275

学院 智能科学与工程学院 系 人工智能 专业 人工智能

实验时间 年 月 日 午～ 月 日 午 温度 ℃湿度

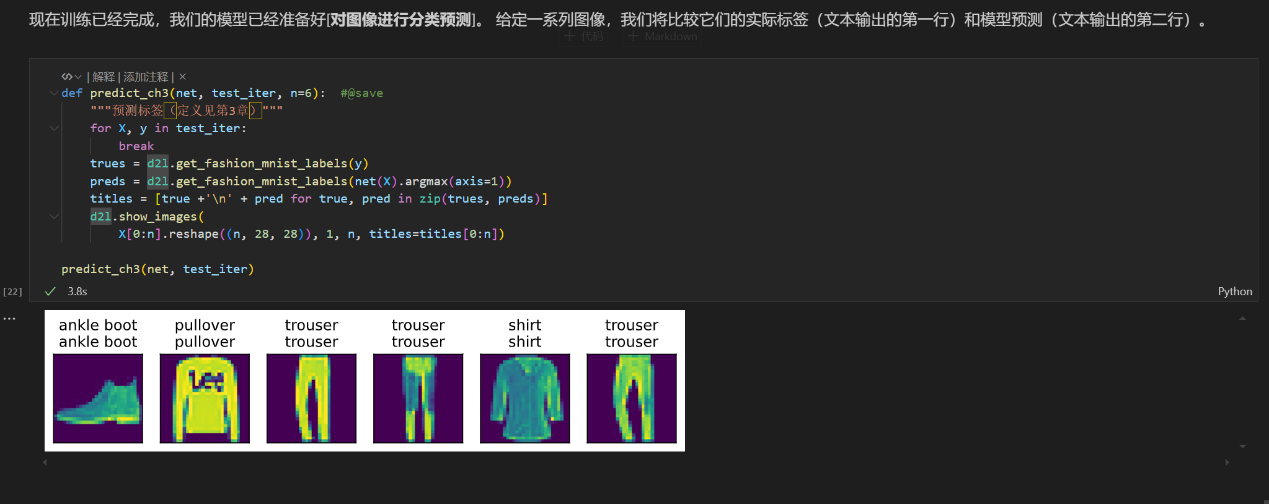
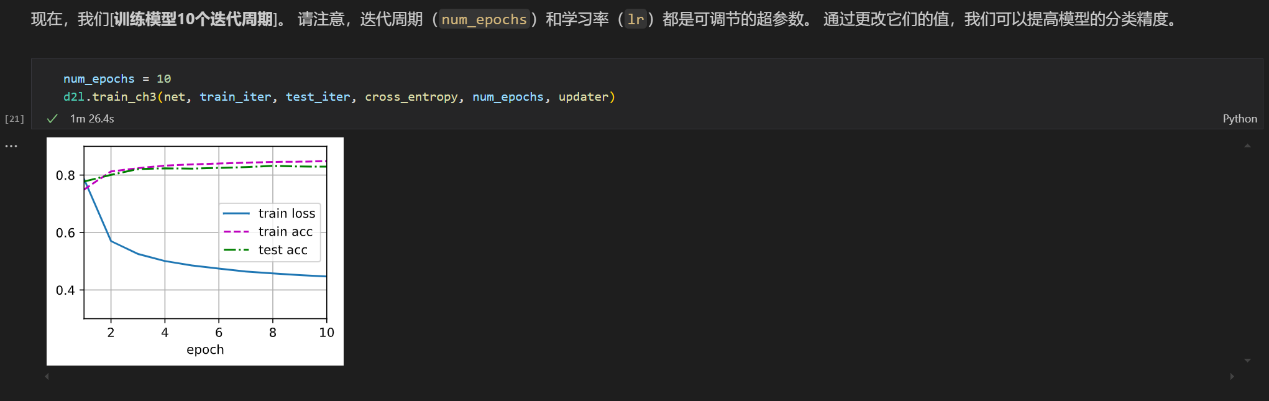
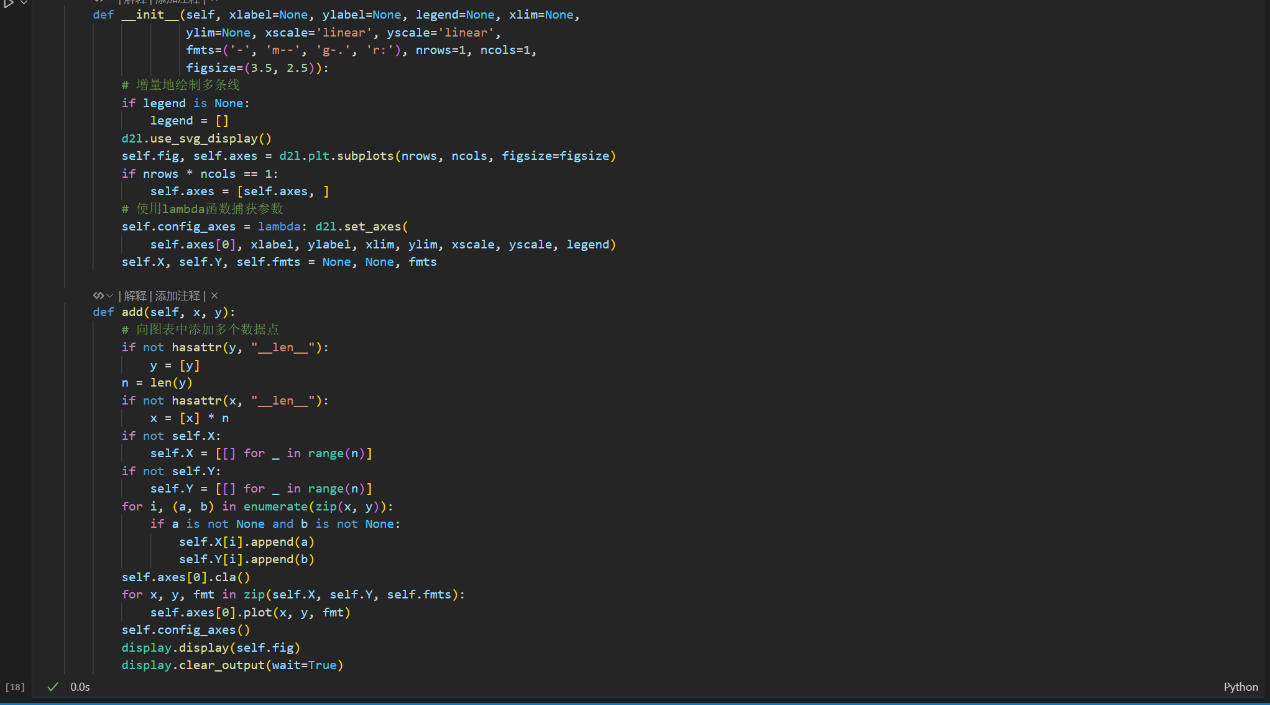
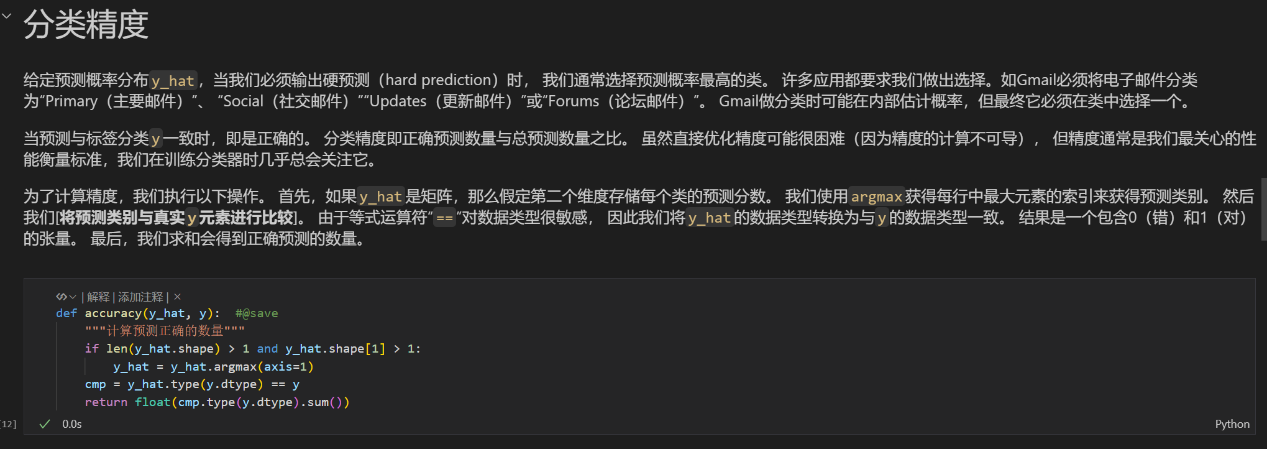
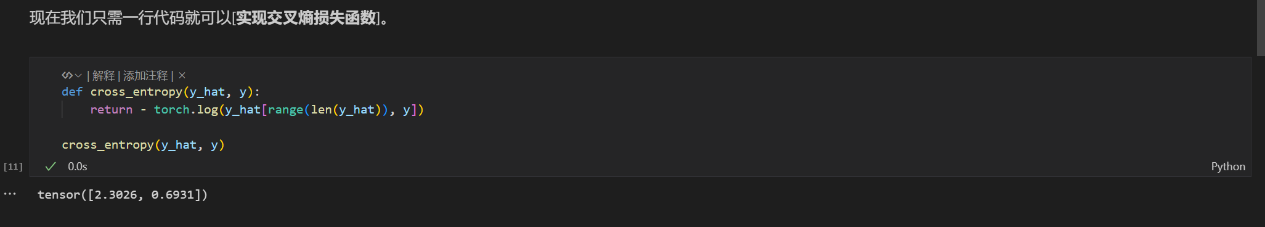
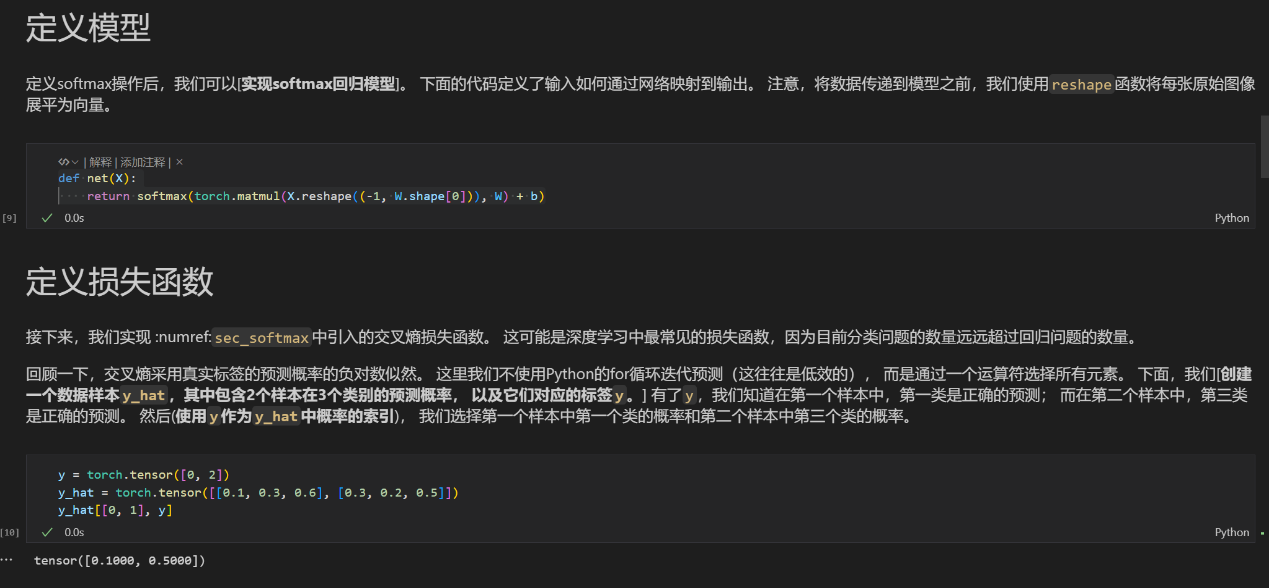
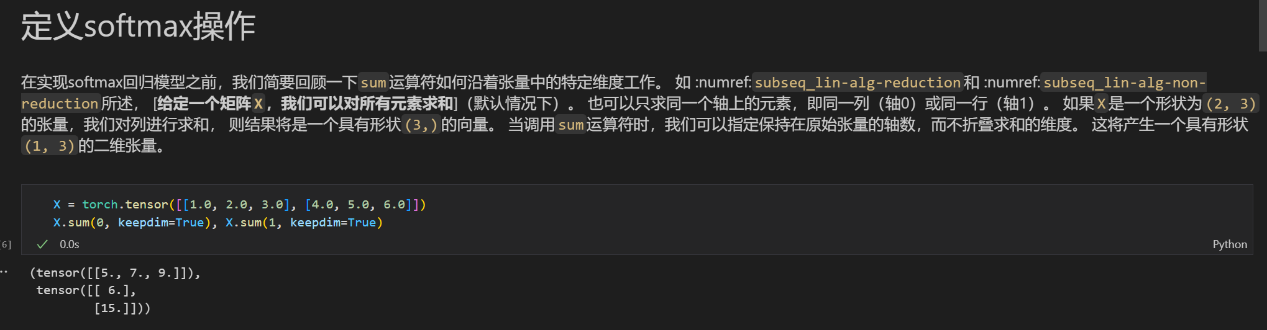
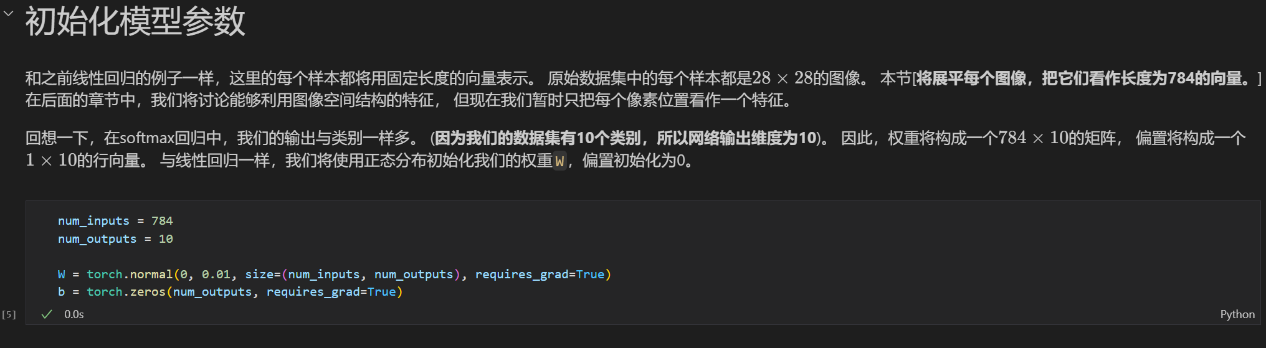
实验内容: 练习 Softmax 回归与 K 折交叉验证.

实验过程:

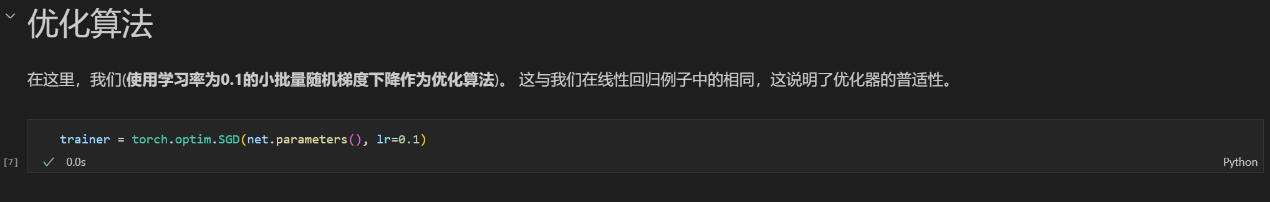
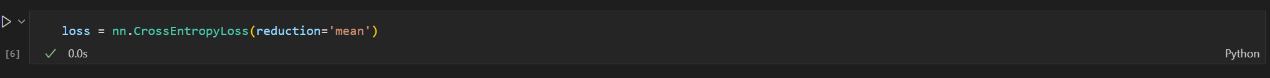
1. 运行” Softmax 回归实现”和” K 折交叉验证”相关章节的 notebook 代码和阅读理论内容.

运行结果如下图：

（一）softmax回归从零开始实现



（二）softmax回归简洁实现

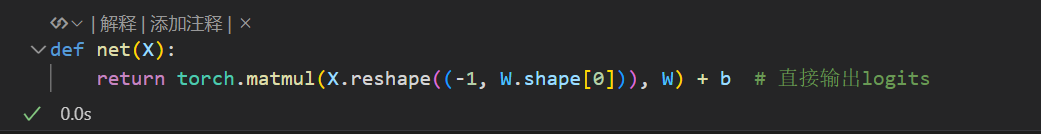
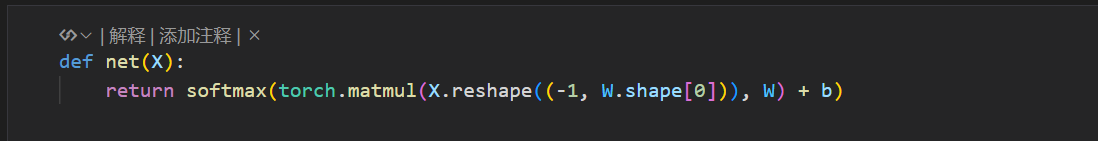


（三）K折交叉验证相关

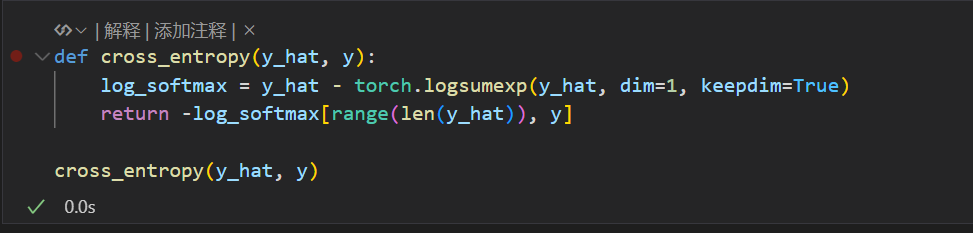
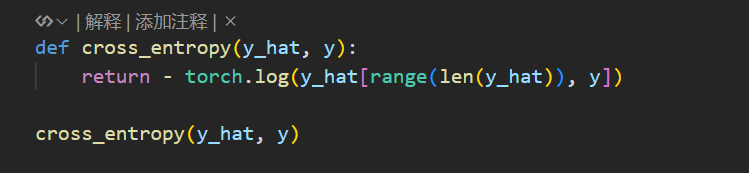
2. 调整 Softmax 层的位置. 将其从网络模型中移动到损失函数中, 完成训练.对比与未调整前模型的准确性性能.

（一）修改处：

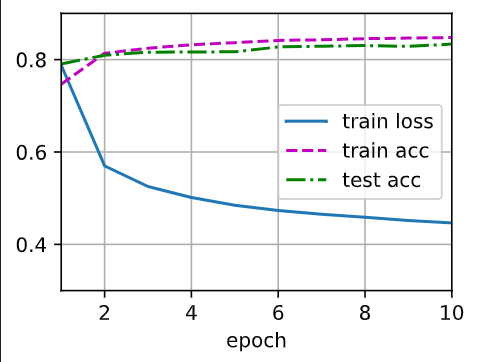
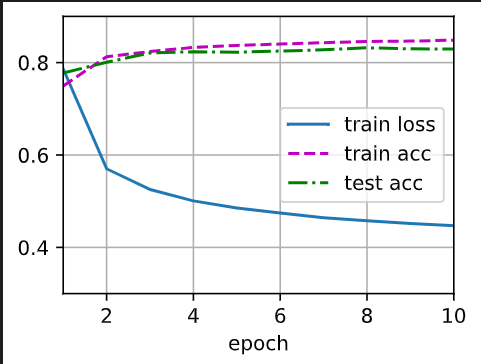
(1) 网络结构调整：原网络最后一层会输出经过softmax的概率分布，修改后直接输出线性层的原始结果（logits）



（2）损失函数重构：原损失函数直接接收概率值，修改后在损失函数内部先计算softmax



（二）结果对比



修改前 修改后

两种实现方式在数学公式上是完全等价的。交叉熵损失结合Softmax的梯度计算与直接对logits使用交叉熵的梯度计算在理论上应产生相同的更新方向。差异主要来自数值稳定性和实现细节，而非算法本身。在Fashion-MNIST数据集上的典型表现：

原模型：最终测试精度约 82%~84%

修改后：测试精度可达 83%~85%（因稳定性的微小提升）

二者在相同稳定实现下应表现一致。

3.使用 Fashion MNIST 数据集, 编写数据加载代码, 实现一个 K 折数据加载器. 预设了 K 时, 可选某一折数据作为验证集. 编写对 K 折的训练-测试的 for循环, 自行设计一个简单的网络模型, 实现交叉验证, 列出每一折的准确率数据.

代码如下：

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88 | **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    # 执行K折交叉验证  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}%") |

功能说明：

我们使用PyTorch的Subset创建K折数据子集，保持原始测试集不变用于最终评估，确认读取为fashion mnist数据集。使用包含两个隐藏层的全连接网络用于训练，使用ReLU激活函数。使用sklearn的KFold进行数据划分，每折独立训练新模型，每折训练10个epoch。使用Adam优化器优化训练。损失函数为交叉熵损失函数，每折训练完成后计算验证集准确率输出结果：显示每折验证准确率。

结果展示：

