

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 °C 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

作为一个从零开始的实现，我们使用 `nnfrefsec_linear_scratch` 中定义的 (小批量随机梯度下降优化模型的损失函数，设置学习率为0.1。

```
lr = 0.1
def updater(batch_size):
    return d2l.sgd(W, b, lr, batch_size)
```

现在，我们训练模型10个迭代周期。请注意，迭代周期 (`num_epochs`) 和学习率 (`lr`) 都是可调节的超参数。通过更改它们的值，我们可以提高模型的分类精度。

```
num_epochs = 10
train_ch1(net, train_loader, test_loader, cross_entropy, num_epochs, updater)
```

预测

现在训练已经完成，我们的模型已经准备好对图像进行分类预测。给定一系列图像，我们将比较它们的实际标签 (文本输出的第一行) 和模型预测 (文本输出的第二行)。

b) 预测

预测

现在训练已经完成，我们的模型已经准备好对图像进行分类预测。给定一系列图像，我们将比较它们的实际标签 (文本输出的第一行) 和模型预测 (文本输出的第二行)。

```
def predict_ch1(net, test_loader, n=6):
    """预测指定数量的图像"""
    for X, y in test_loader:
        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[:n].reshape((n, 28, 28)), 1, n, titles=titles[:n])
    predict_ch1(net, test_loader)
```

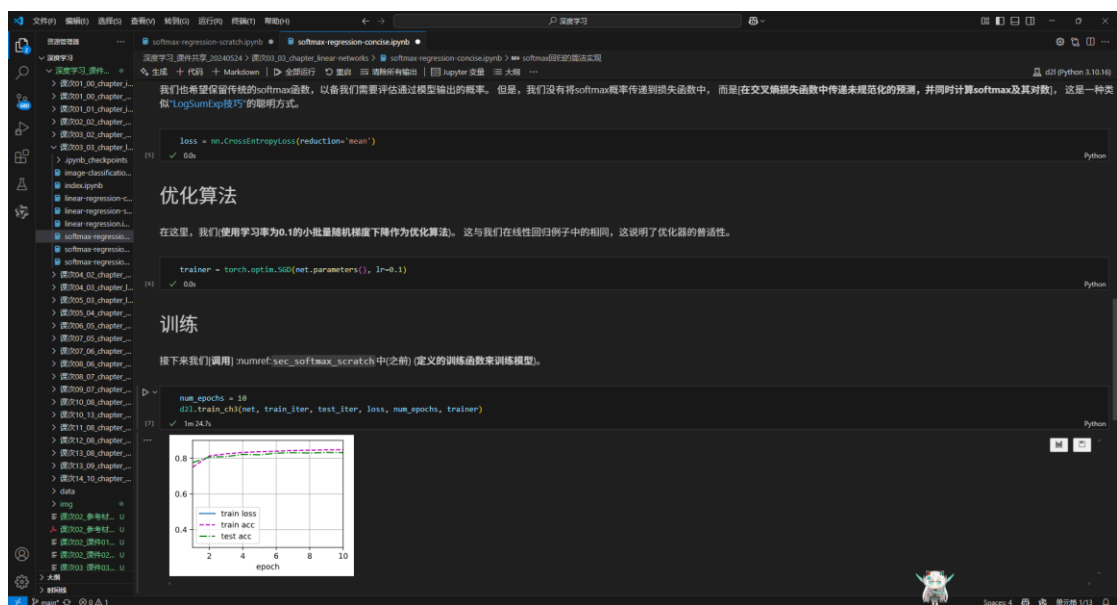
小结

- 借助softmax回归，我们可以训练多分类的模型。
- 训练softmax回归循环模型与训练线性回归模型非常相似：先读取数据，再定义模型和损失函数，然后使用优化算法训练模型。大多数常见的深度学习模型都有类似的训练过程。

练习

1. 本节直接实现了基于数学定义softmax运算的 `softmax` 函数。这可能会导致什么问题？提示：尝试计算 `exp(50)` 的大小。

(2) softmax 回归的简洁实现



2. 调整 Soffmax 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 softmax,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

定义损失函数

接下来，我们实现 `nn.CrossEntropyLoss` 中引入的交叉熵损失函数。这可能是深度学习中最常见的损失函数，因为目前分类问题的数量远远超过回归问题的数量。

回顾一下，交叉熵采用真实标签的预测概率的负对数似然。这里我们不使用Python的for循环迭代预测（这往往是低效的），而是通过一个运算符选择所有元素。下面，我们创建一个数据样本 `y_hat`，其中包含2个样本在3个类别的预测概率，以及它们对应的标签 `y`。有了 `y`，我们知道在第一个样本中，第一类是正确的预测；而在第二个样本中，第三类是正确的预测。然后使用 `y` 作为 `y_hat` 中概率的索引，我们选择第一个样本中第一个类的概率和第二个样本中第三个类的概率。

```
y = torch.tensor([0, 2])
y_hat = torch.tensor([[0.1, 0.3, 0.6], [0.3, 0.2, 0.5]])
y_hat[[0, 1], y]
# 0.6
tensor([0.1000, 0.5000])

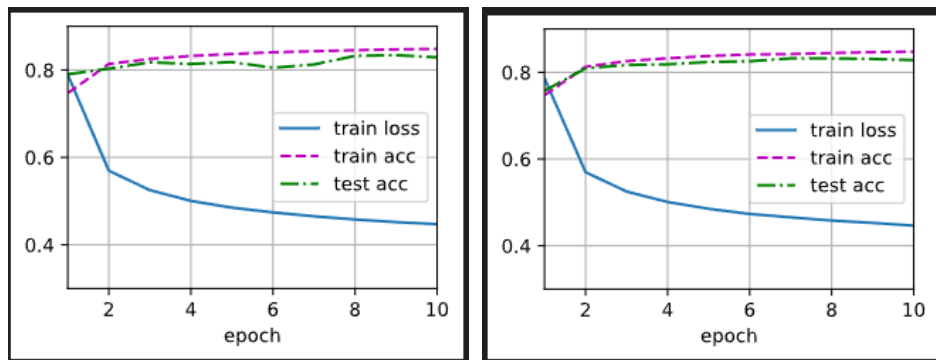
现在只需一行代码就可以实现交叉熵损失函数。

def cross_entropy(y_hat, y):
    # 先进行softmax操作
    y_hat = softmax(y_hat)
    return - torch.log(y_hat[range(len(y_hat)), y])

cross_entropy(y_hat, y)
# 0.6
tensor([1.3533, 0.9398])
```

(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

预测

现在训练已经完成，我们的模型已经准备好对图像进行分类预测。给定一系列图像，我们将比较它们的实际标签（文本输出的第一行）和模型预测（文本输出的第二行）。

```
def predict_ch3(net, test_iter, n=6): #@save
    """预测标签(见第3章)"""
    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])

predict_ch3(net, test_iter)
```

13%

ankle boot	pullover	trouser	trouser	shirt	trouser
ankle boot	pullover	trouser	trouser	shirt	trouser

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）

```

# 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}%")

```

Output:

```

Fold 1
Fold 1 Validation Accuracy: 88.60%
Fold 2
Fold 2 Validation Accuracy: 89.33%
Fold 3
Fold 3 Validation Accuracy: 89.79%
Fold 4
Fold 4 Validation Accuracy: 89.38%
Fold 5
Fold 5 Validation Accuracy: 89.36%
K-Fold Cross Validation Results:
Fold 1: 88.60%
Fold 2: 89.33%
Fold 3: 89.79%
Fold 4: 89.38%
Fold 5: 89.36%
Average Accuracy: 89.29%

```