# 使用交叉熵实现一个多分类任务

## 目标

五位随机向量最大的数字在哪维就属于哪一类。

## 修改内容

### 模型结构

1. 输出层改为5个神经元（对应5个类别）
2. Loss function 改为nn.CrossEntropyLoss()

### 数据生成

1. Build\_sample()函数返回最大值索引作为类别标签（0-4）
2. 标签类型从Float改为Long（交叉熵要求）

### 评估函数

1. 使用torch.argmax()获取预测类别
2. 直接比较预测类别与真实类别计算准确率。

### 训练过程

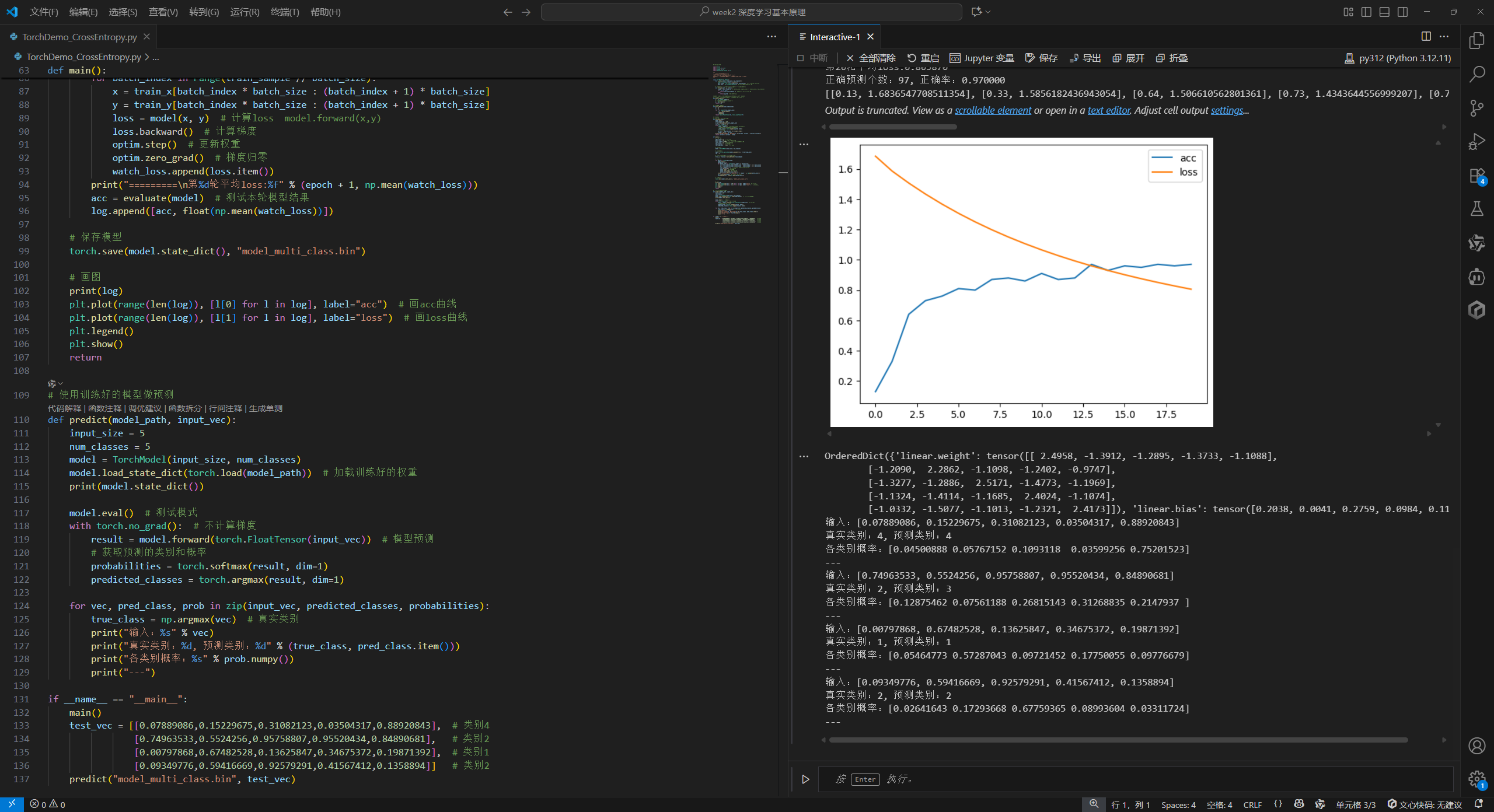
1. 标签格式适配交叉熵损失函数要求。
2. 保持原有的训练流程。

### 预测函数

1. 添加了概率计算（使用softmax）。
2. 显示各类别的预测概率。

## 测试

### 部署+运行



### 本地代码路径

Z:\BaDou\预习视频-录播\第二周 深度学习原理\_0902145000\week2 深度学习基本原理\week2 深度学习基本原理\TorchDemo\_CrossEntropy.py

### 源代码

# coding:utf8

import torch

import torch.nn as nn

import numpy as np

import matplotlib.pyplot as plt

"""

基于pytorch框架编写模型训练

实现一个自行构造的找规律(机器学习)任务

规律：x是一个5维向量，最大值所在的索引位置为类别标签(0-4)

"""

class TorchModel(nn.Module):

    def \_\_init\_\_(self, input\_size, num\_classes=5):

        super(TorchModel, self).\_\_init\_\_()

        self.linear = nn.Linear(input\_size, num\_classes)  # 线性层，输出5个类别

        self.loss = nn.CrossEntropyLoss()  # loss函数采用交叉熵损失

    # 当输入真实标签，返回loss值；无真实标签，返回预测值

    def forward(self, x, y=None):

        y\_pred = self.linear(x)  # (batch\_size, input\_size) -> (batch\_size, num\_classes)

        if y is not None:

            return self.loss(y\_pred, y)  # 预测值和真实值计算损失

        else:

            return y\_pred  # 输出预测结果

# 生成一个样本, 样本的生成方法，代表了我们要学习的规律

# 随机生成一个5维向量，最大值所在的索引位置为类别标签

def build\_sample():

    x = np.random.random(5)

    # 找到最大值的索引作为类别标签

    y = np.argmax(x)

    return x, y

# 随机生成一批样本

def build\_dataset(total\_sample\_num):

    X = []

    Y = []

    for i in range(total\_sample\_num):

        x, y = build\_sample()

        X.append(x)

        Y.append(y)

    return torch.FloatTensor(X), torch.LongTensor(Y)

# 测试代码

# 用来测试每轮模型的准确率

def evaluate(model):

    model.eval()

    test\_sample\_num = 100

    x, y = build\_dataset(test\_sample\_num)

    correct, wrong = 0, 0

    with torch.no\_grad():

        y\_pred = model(x)  # 模型预测 model.forward(x)

        # 获取预测的类别（最大概率的索引）

        y\_pred\_class = torch.argmax(y\_pred, dim=1)

        # 计算准确率

        correct = (y\_pred\_class == y).sum().item()

        wrong = len(y) - correct

    print("正确预测个数：%d, 正确率：%f" % (correct, correct / (correct + wrong)))

    return correct / (correct + wrong)

def main():

    # 配置参数

    epoch\_num = 20  # 训练轮数

    batch\_size = 20  # 每次训练样本个数

    train\_sample = 5000  # 每轮训练总共训练的样本总数

    input\_size = 5  # 输入向量维度

    num\_classes = 5  # 分类数

    learning\_rate = 0.001  # 学习率

    # 建立模型

    model = TorchModel(input\_size, num\_classes)

    # 选择优化器

    optim = torch.optim.Adam(model.parameters(), lr=learning\_rate)

    log = []

    # 创建训练集，正常任务是读取训练集

    train\_x, train\_y = build\_dataset(train\_sample)

    # 训练过程

    for epoch in range(epoch\_num):

        model.train()

        watch\_loss = []

        for batch\_index in range(train\_sample // batch\_size):

            x = train\_x[batch\_index \* batch\_size : (batch\_index + 1) \* batch\_size]

            y = train\_y[batch\_index \* batch\_size : (batch\_index + 1) \* batch\_size]

            loss = model(x, y)  # 计算loss  model.forward(x,y)

            loss.backward()  # 计算梯度

            optim.step()  # 更新权重

            optim.zero\_grad()  # 梯度归零

            watch\_loss.append(loss.item())

        print("=========\n第%d轮平均loss:%f" % (epoch + 1, np.mean(watch\_loss)))

        acc = evaluate(model)  # 测试本轮模型结果

        log.append([acc, float(np.mean(watch\_loss))])

    # 保存模型

    torch.save(model.state\_dict(), "model\_multi\_class.bin")

    # 画图

    print(log)

    plt.plot(range(len(log)), [l[0] for l in log], label="acc")  # 画acc曲线

    plt.plot(range(len(log)), [l[1] for l in log], label="loss")  # 画loss曲线

    plt.legend()

    plt.show()

    return

# 使用训练好的模型做预测

def predict(model\_path, input\_vec):

    input\_size = 5

    num\_classes = 5

    model = TorchModel(input\_size, num\_classes)

    model.load\_state\_dict(torch.load(model\_path))  # 加载训练好的权重

    print(model.state\_dict())

    model.eval()  # 测试模式

    with torch.no\_grad():  # 不计算梯度

        result = model.forward(torch.FloatTensor(input\_vec))  # 模型预测

        # 获取预测的类别和概率

        probabilities = torch.softmax(result, dim=1)

        predicted\_classes = torch.argmax(result, dim=1)

    for vec, pred\_class, prob in zip(input\_vec, predicted\_classes, probabilities):

        true\_class = np.argmax(vec)  # 真实类别

        print("输入：%s" % vec)

        print("真实类别：%d, 预测类别：%d" % (true\_class, pred\_class.item()))

        print("各类别概率：%s" % prob.numpy())

        print("---")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

    test\_vec = [[0.07889086,0.15229675,0.31082123,0.03504317,0.88920843],  # 类别4

                [0.74963533,0.5524256,0.95758807,0.95520434,0.84890681],   # 类别2

                [0.00797868,0.67482528,0.13625847,0.34675372,0.19871392],  # 类别1

                [0.09349776,0.59416669,0.92579291,0.41567412,0.1358894]]   # 类别2

    predict("model\_multi\_class.bin", test\_vec)