**实 验 报 告**

**学 院：** 计算机科学与技术学院

**班 级：** 计非211

**学 号：** 202131990522

**姓 名：** 嵇纹垲

**课 程：** 人工智能基础

**指导教师：** 周昌军

**完成时间：** 2024 **年** 4 **月** 16 **日**

**浙 江 师 范 大 学 制**

**实验三：神经网络实现鸢尾花分类实验要求**

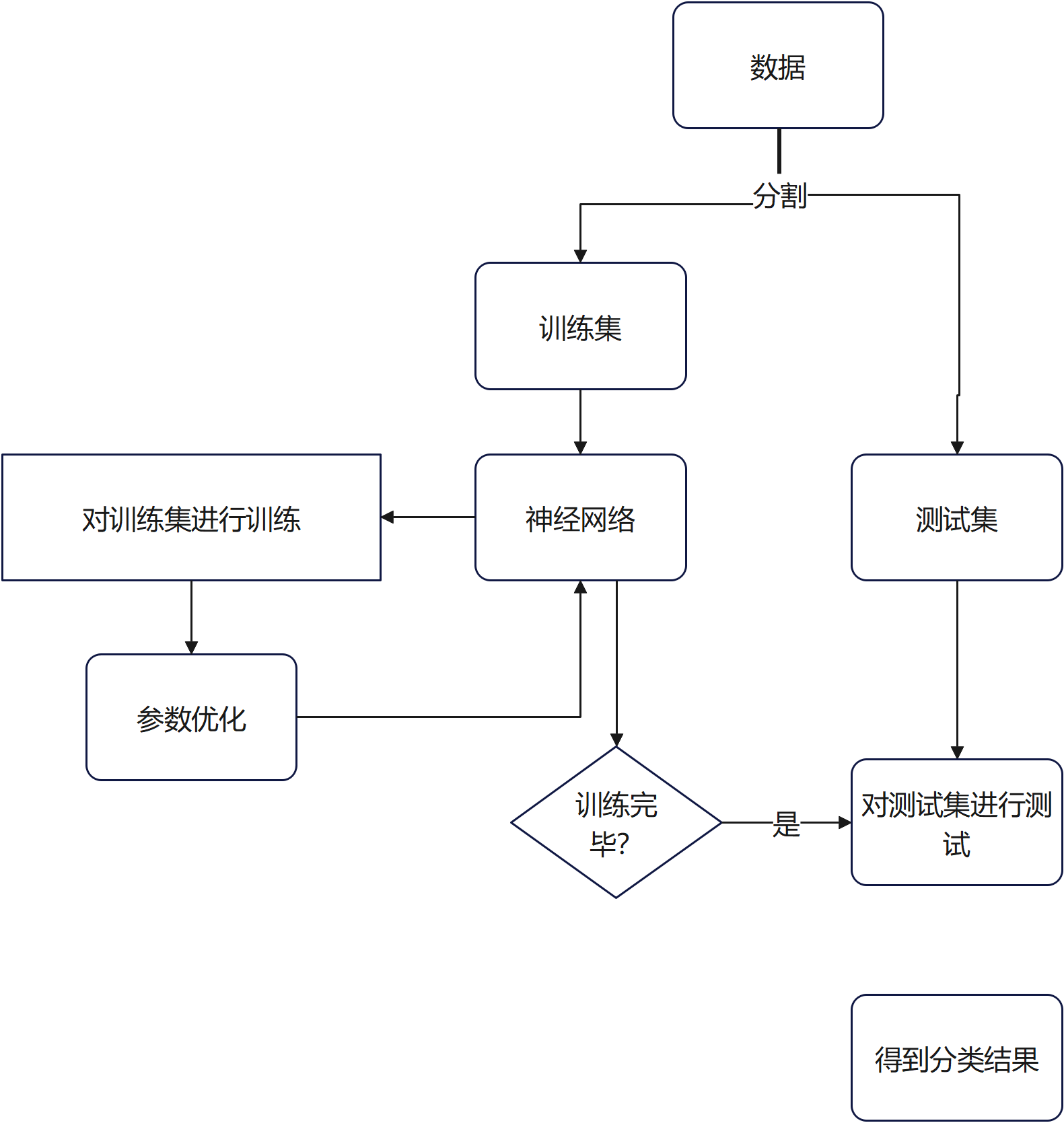
**实验题目：**

用神经网络解决鸢尾花分类的问题

**问题分析：**

鸢尾花的数据含有四个变量，分别为花萼的长、宽，以及花瓣的长、宽；而其种类一共有三种。所以我们需要做的工作是将一个四维向量对应到一个三种分类中的一种。

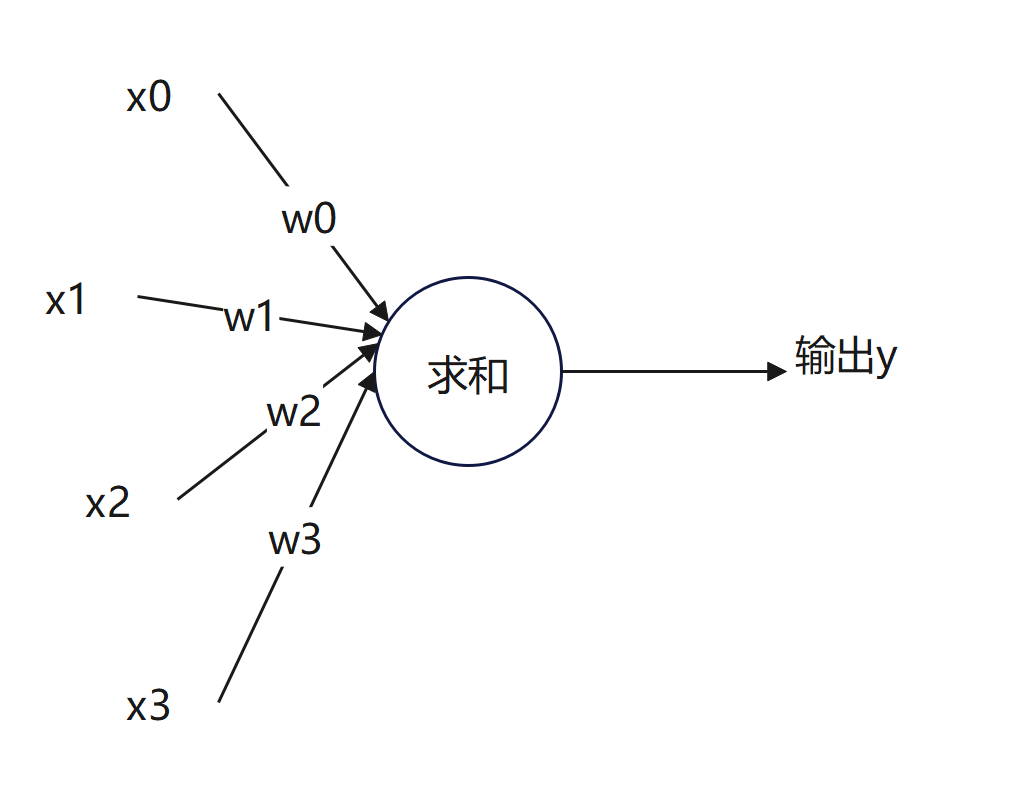
由此我们可以采用神经网络来解决该问题。



**算法设计：**

使用神经网络来处理分类问题，我们首先需要一个建立一个简易的网络。

如图，

****

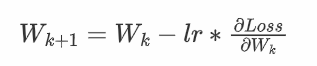
对于每一个数据，我们对输入的向量x的每一维乘上一个权值w，最后再加上偏置项b，求和并得到最终的值y。然后根据其划归到各个分类的概率即可确定该组数据的最终分类是什么。

为此，我们需要找到最佳的神经网络的权值来尽可能的降低预测值的误差，可以采用梯度下降法。

首先定义损失函数Loss，它可以定量判断w和b参数选择的优劣，我们的目的就是要让Loss函数的值尽可能的小。可以采用常见的均方误差。损失函数的梯度表示损失函数对各参数求偏导后的向量，损失函数梯度下降的方向，就是是损失函数减小的方向。梯度下降法即沿着损失函数梯度下降的方向，寻找损失函数的最小值，从而得到最优的参数。

每次如果沿着函数的梯度不断下降的话，最终我们就可以来到一个局部最优解。如果初始位置随机的次数足够多，我们最终就可以足够接近全局最优解。

为此，每次训练完之后，我们对W\_{k}={w1\_{k},w2\_{k},...}，也就是所有权值组成的向量进行自更新，沿着梯度下降。当然，每次下降的步长lr需要我们自己调整。

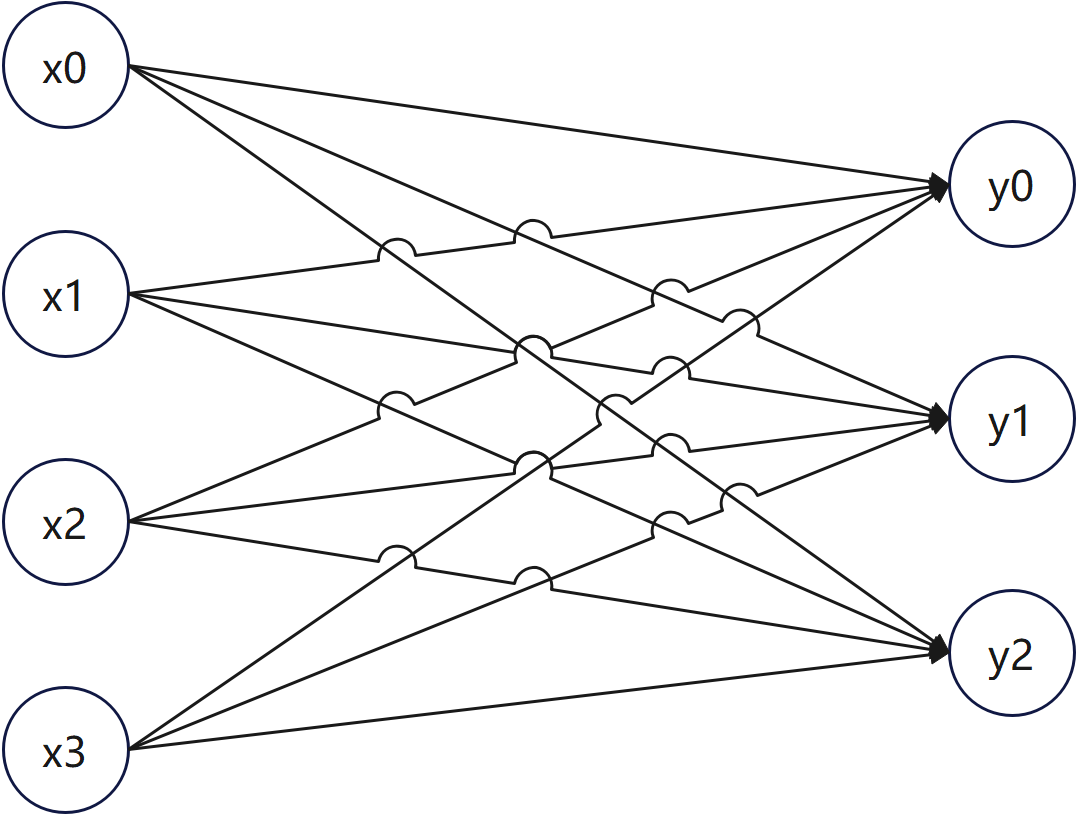


更新完网络的权值之后，再通过计算得到y即可



注意到权值会自更新，所以一开始初始权值我们可以随机附一个值。

最终我们就可以得到一个输入层大小为4，输出层大小为3的全连接神经网络

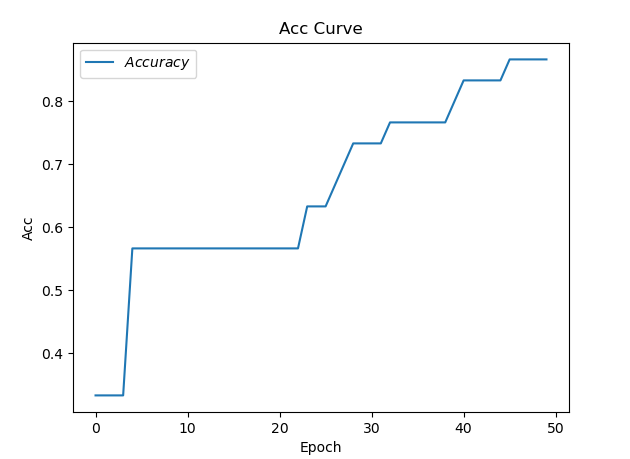
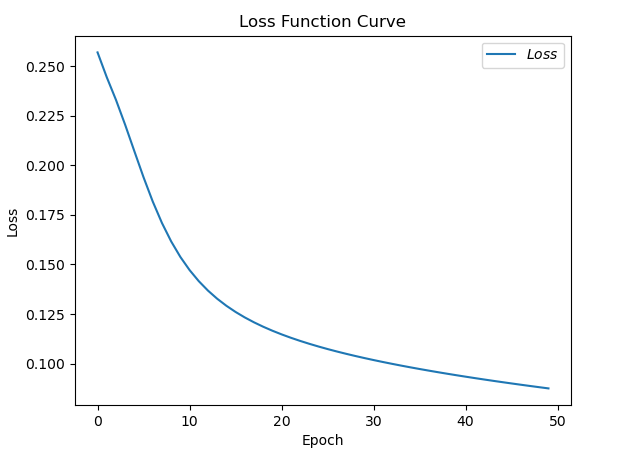


但是对于误差函数Loss的计算，考虑到分类实际上是一种离散变量，所以为了保证不同分类两两两之间的距离完全等价，我们可以采用独热编码，从而可以通过欧式距离来直接计算距离。

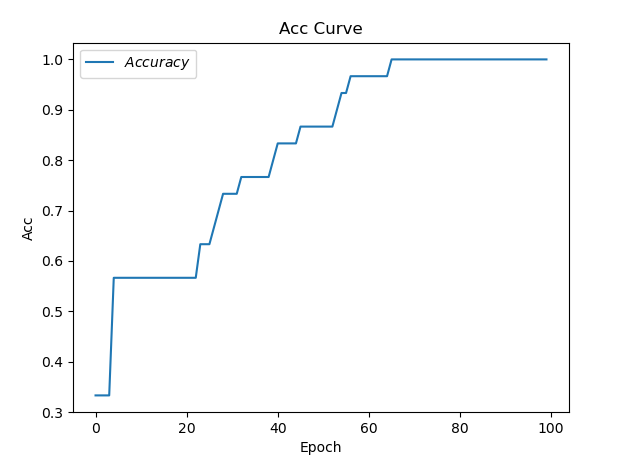
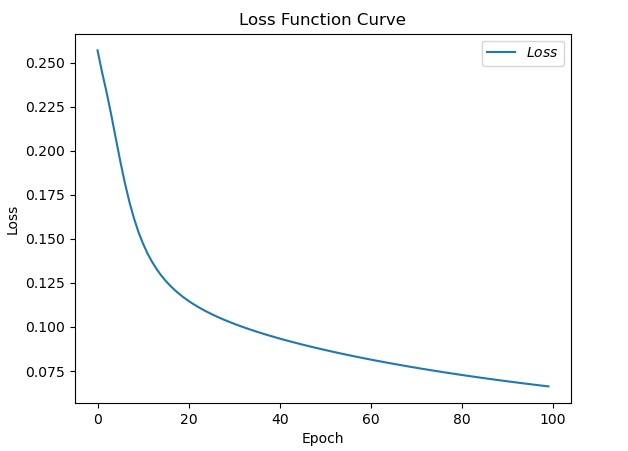
**运算结果：**

以下展示训练50/100/500次的损失函数和分类准确率的变化

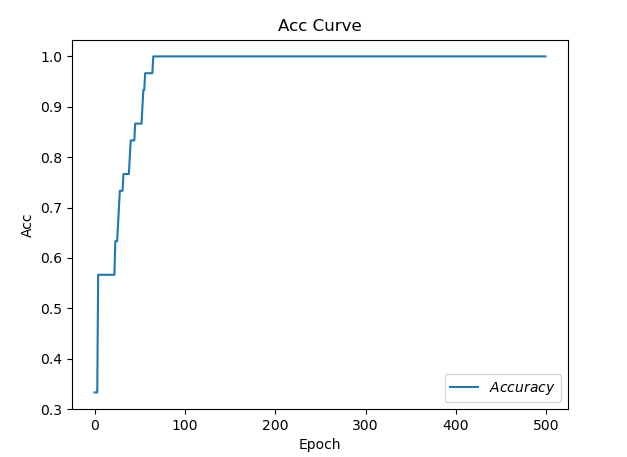
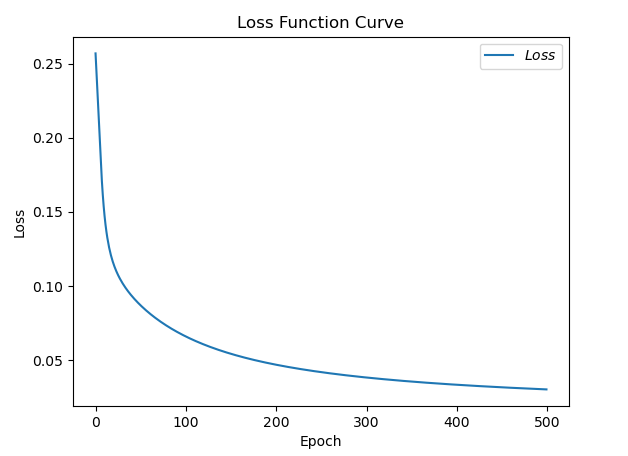
训练50次的结果



训练100次的结果



训练500次的结果



可以看到，随着预测次数的增加，损失函数确实呈现了下降的趋势，而相对应的，分类的准确率也在不断上升。但是当预测次数上升到一定程度的时候，两者都不再呈现明显的变化。

**实验总结：**

本次实验利用神经网络对鸢尾花进行分类，采用了梯度下降算法对网络参数进行优化。并且使用了独热编码，将分类标签转换为适合神经网络处理的格式。最终，模型在测试集上达到了接近100%的准确率。实验结果表明，神经网络能够有效识别鸢尾花的种类，具有较高的分类准确性和泛化能力。同时，该实验为进一步探索神经网络在分类问题中的应用提供了有益的参考。

**代码：**

from sklearn import datasets

import numpy as np

from matplotlib import pyplot as plt

import tensorflow as tf

import os

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'

# 导入数据

x\_data = datasets.load\_iris().data

y\_data = datasets.load\_iris().target

# 打乱

np.random.seed(2106) # 使用相同的seed，保证输入特征和标签一一对应

np.random.shuffle(x\_data)

np.random.seed(2106)

np.random.shuffle(y\_data)

tf.random.set\_seed(2106)

# for i in range(0,len(x\_data)):

# print(x\_data[i],y\_data[i])

# 将数据分为训练集和数据集

x\_train = x\_data[:-30]

y\_train = y\_data[:-30]

x\_test = x\_data[-30:]

y\_test = y\_data[-30:]

# 将数据变成矩阵类型

x\_train = tf.cast(x\_train, tf.float32)

x\_test = tf.cast(x\_test, tf.float32)

# 将数据与对应标签整合，每组32个数据

train\_db = tf.data.Dataset.from\_tensor\_slices((x\_train, y\_train)).batch(32)

test\_db = tf.data.Dataset.from\_tensor\_slices((x\_test, y\_test)).batch(32)

# 生成神经网络的参数，4个输入特征故，输入层为4个输入节点；因为3分类，故输出层为3个神经元

w1 = tf.Variable(tf.random.truncated\_normal([4, 3], stddev=0.1))

b1 = tf.Variable(tf.random.truncated\_normal([3], stddev=0.1))

# 一开始的边权是随机的，后面在训练过程中会加以更新

lr = 0.1 # 学习率为0.1，也就是梯度下降时的步长

train\_loss\_results = [] # 将每轮的loss记录在此列表中，为后续画loss曲线提供数据

test\_acc = [] # 将每轮的acc记录在此列表中，为后续画acc曲线提供数据

epoch = 500 # 循环轮数

loss\_all = 0 # 每轮分4个step，loss\_all记录四个step生成的4个loss的和

# 训练

for epoch in range(epoch): # 数据集级别的循环，每个epoch循环一次数据集

for step, (x\_train, y\_train) in enumerate(train\_db): # batch级别的循环 ，每个step循环一个batch

# print(step,type(x\_train),len(y\_train))

with tf.GradientTape() as tape: # with结构记录梯度信息

y = tf.matmul(x\_train, w1) + b1 # 神经网络运算

y = tf.nn.softmax(y) # 使输出y符合概率分布（此操作后与独热码同量级，可相减求loss）

# print(y)

y\_ = tf.one\_hot(y\_train, depth=3) # 将标签值转换为独热码格式，方便计算loss和accuracy

# 采用均方误差损失函数mse = mean(sum(y-out)^2)

loss = tf.reduce\_mean(tf.square(y\_ - y))

loss\_all += loss.numpy() # 将每个step计算出的loss累加，为后续求loss平均值提供数据，这样计算的loss更准确

# 计算loss对各个参数的梯度

grads = tape.gradient(loss, [w1, b1])

# 实现梯度更新 w1 = w1 - lr \* w1\_grad b = b - lr \* b\_grad

w1.assign\_sub(lr \* grads[0]) # 参数w1自更新

b1.assign\_sub(lr \* grads[1]) # 参数b自更新

# 每个epoch，打印loss信息

print("Epoch {}, loss: {}".format(epoch, loss\_all / 4))

train\_loss\_results.append(loss\_all / 4) # 将4个step的loss求平均记录在此变量中

loss\_all = 0 # loss\_all归零，为记录下一个epoch的loss做准备

# 测试部分

# total\_correct为预测对的样本个数, total\_number为测试的总样本数，将这两个变量都初始化为0

total\_correct, total\_number = 0, 0

for x\_test, y\_test in test\_db:

# 使用更新后的参数进行预测

y = tf.matmul(x\_test, w1) + b1

y = tf.nn.softmax(y)

pred = tf.argmax(y, axis=1) # 返回y中最大值的索引，即预测的分类

# 将pred转换为y\_test的数据类型

pred = tf.cast(pred, dtype=y\_test.dtype)

# 若分类正确，则correct=1，否则为0，将bool型的结果转换为int型

correct = tf.cast(tf.equal(pred, y\_test), dtype=tf.int32)

# 将每个batch的correct数加起来

correct = tf.reduce\_sum(correct)

# 将所有batch中的correct数加起来

total\_correct += int(correct)

# total\_number为测试的总样本数，也就是x\_test的行数，shape[0]返回变量的行数

total\_number += x\_test.shape[0]

# 总的准确率等于total\_correct/total\_number

acc = total\_correct / total\_number

test\_acc.append(acc)

print("Test\_acc:", acc)

print("--------------------------")

# 绘制 loss 曲线

plt.title('Loss Function Curve') # 图片标题

plt.xlabel('Epoch') # x轴变量名称

plt.ylabel('Loss') # y轴变量名称

# 逐点画出trian\_loss\_results值并连线，连线图标是Loss

plt.plot(train\_loss\_results, label="$Loss$")

plt.legend() # 画出曲线图标

plt.show() # 画出图像

# 绘制 Accuracy 曲线

plt.title('Acc Curve') # 图片标题

plt.xlabel('Epoch') # x轴变量名称

plt.ylabel('Acc') # y轴变量名称

plt.plot(test\_acc, label="$Accuracy$") # 逐点画出test\_acc值并连线，连线图标是Accuracy

plt.legend()

plt.show()