**实验（实习）报告**

1. 实验目的

通过实验了解全连接神经网络的结构，应用全连接网络处理分类和回归任务。

1. 实验任务

1）初级实验：①手写体图像识别实验；②FashionMnist图像分类实验；③汽车里程数预测实验。

2）中级实验：①鸢尾花分类任务对比实验。

1. 实验步骤
2. 手写体图像识别
   1. 导入实验所需模块

#导入相关依赖库

import  os

import numpy as np

from matplotlib import pyplot as plt

import mindspore as ms

#context模块用于设置实验环境和实验设备

import mindspore.context as context

#dataset模块用于处理数据形成数据集

import mindspore.dataset as ds

#c\_transforms模块用于转换数据类型

import mindspore.dataset.transforms.c\_transforms as C

#vision.c\_transforms模块用于转换图像，这是一个基于opencv的高级API

import mindspore.dataset.vision.c\_transforms as CV

#导入Accuracy作为评价指标

from mindspore.nn.metrics import Accuracy

#nn中有各种神经网络层如：Dense，ReLu

from mindspore import nn

#Model用于创建模型对象，完成网络搭建和编译，并用于训练和评估

from mindspore.train import Model

#LossMonitor可以在训练过程中返回LOSS值作为监控指标

from mindspore.train.callback import  LossMonitor

#设定运行模式为动态图模式，并且运行设备为昇腾芯片

context.set\_context(mode=context.GRAPH\_MODE, device\_target='Ascend')

* 1. 导入实验数据集

MNIST是一个手写数字数据集，训练集包含60000张手写数字，测试集包含10000张手写数字，共10类。可在MNIST数据集的官网下载数据集，解压到当前代码目录下。MindSpore的dataset模块有专门用于读取和解析MNIST数据集的源数据集，可直接读取并生成训练集和测试集。

步骤 1加载并查看数据集

#MindSpore内置方法读取MNIST数据集

ds\_train = ds.MnistDataset(os.path.join(r'./MNIST', "train"))

ds\_test = ds.MnistDataset(os.path.join(r'./MNIST', "test"))

print('训练数据集数量：',ds\_train.get\_dataset\_size())

print('测试数据集数量：',ds\_test.get\_dataset\_size())

#该数据集可以通过create\_dict\_iterator()转换为迭代器形式，然后通过\_\_next\_\_()一个个输出样本

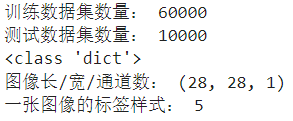
image=ds\_train.create\_dict\_iterator().\_\_next\_\_()

print(type(image))

print('图像长/宽/通道数：',image['image'].shape)

#一共10类，用0-9的数字表达类别。

print('一张图像的标签样式：',image['label'])



步骤 2生成测试集和训练集

DATA\_DIR\_TRAIN = "./MNIST/train" # 训练集信息

DATA\_DIR\_TEST = "./MNIST/test" # 测试集信息

def create\_dataset(training=True, batch\_size=128, resize=(28, 28),rescale=1/255, shift=-0.5, buffer\_size=64):

    ds = ms.dataset.MnistDataset(DATA\_DIR\_TRAIN if training else DATA\_DIR\_TEST)

    #定义改变形状、归一化和更改图片维度的操作。

    #改为（28,28）的形状

    resize\_op = CV.Resize(resize)

    #rescale方法可以对数据集进行归一化和标准化操作，这里就是将像素值归一到0和1之间，shift参数可以让值域偏移至-0.5和0.5之间

    rescale\_op = CV.Rescale(rescale, shift)

    #由高度、宽度、深度改为深度、高度、宽度

    hwc2chw\_op = CV.HWC2CHW()

    # 利用map操作对原数据集进行调整

    ds = ds.map(input\_columns="image", operations=[resize\_op, rescale\_op, hwc2chw\_op])

    ds = ds.map(input\_columns="label", operations=C.TypeCast(ms.int32))

    #设定洗牌缓冲区的大小，从一定程度上控制打乱操作的混乱程度

    ds = ds.shuffle(buffer\_size=buffer\_size)

    #设定数据集的batch\_size大小，并丢弃剩余的样本

    ds = ds.batch(batch\_size, drop\_remainder=True)

    return ds

步骤 3查看数据

#显示前10张图片以及对应标签,检查图片是否是正确的数据集

dataset\_show = create\_dataset(training=False)

data = dataset\_show.create\_dict\_iterator().\_\_next\_\_()

images = data['image'].asnumpy()

labels = data['label'].asnumpy()

for i in range(1,11):

    plt.subplot(2, 5, i)

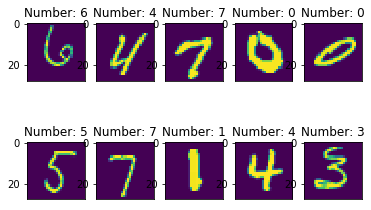
    #利用squeeze方法去掉多余的一个维度

    plt.imshow(np.squeeze(images[i]))

    plt.title('Number: %s' % labels[i])

    plt.xticks([])

plt.show()



* 1. 模型搭建与训练

手写数字图像数据集准备完成，接下来我们就需要构建训练模型，本实验采用的是全连接神经网络算法，所以我们首先需要建立初始化的神经网络。

步骤 1创建网络

#利用定义类的方式生成网络，Mindspore中定义网络需要继承nn.cell。在init方法中定义该网络需要的神经网络层

#在construct方法中梳理神经网络层与层之间的关系。

class ForwardNN(nn.Cell):

    def \_\_init\_\_(self):

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

        self.flatten = nn.Flatten()

        self.relu = nn.ReLU()

        self.fc1 = nn.Dense(784, 512, activation='relu')

        self.fc2 = nn.Dense(512, 256, activation='relu')

        self.fc3 = nn.Dense(256, 128, activation='relu')

        self.fc4 = nn.Dense(128, 64, activation='relu')

        self.fc5 = nn.Dense(64, 32, activation='relu')

        self.fc6 = nn.Dense(32, 10, activation='softmax')

    def construct(self, input\_x):

        output = self.flatten(input\_x)

        output = self.fc1(output)

        output = self.fc2(output)

        output = self.fc3(output)

        output = self.fc4(output)

        output = self.fc5(output)

        output = self.fc6(output)

        return output

步骤 2设定参数

lr = 0.001

num\_epoch = 10

momentum = 0.9

net = ForwardNN()

#定义loss函数，改函数不需要求导，可以给离散的标签值，且loss值为均值

loss = nn.loss.SoftmaxCrossEntropyWithLogits( sparse=True, reduction='mean')

#定义准确率为评价指标，用于评价模型

metrics={"Accuracy": Accuracy()}

#定义优化器为Adam优化器，并设定学习率

opt = nn.Adam(net.trainable\_params(), lr)

步骤 3训练模型并保存

#生成验证集，验证机不需要训练，所以不需要repeat

ds\_eval = create\_dataset(False, batch\_size=32)

#模型编译过程，将定义好的网络、loss函数、评价指标、优化器编译

model = Model(net, loss, opt, metrics)

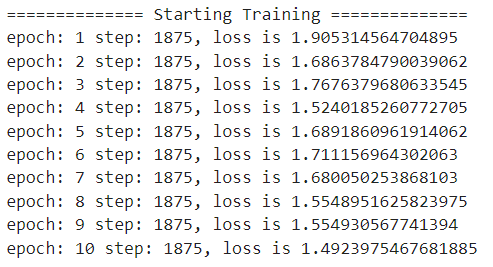
#生成训练集

ds\_train = create\_dataset(True, batch\_size=32)

print("============== Starting Training ==============")

#训练模型，用loss作为监控指标，并利用昇腾芯片的数据下沉特性进行训练

model.train(num\_epoch, ds\_train,callbacks=[LossMonitor()],dataset\_sink\_mode=True)



* 1. 模型评估

利用模型对测试集的数据进行预测，并与标签对比，用准确率accuarcy进行评估。

#使用测试集评估模型，打印总体准确率

metrics\_result=model.eval(ds\_eval)

print(metrics\_result)



1. FashionMnist图像分类实验
   1. 导入实验所需模块

import os

import struct

from easydict import EasyDict as edict

import matplotlib.pyplot as plt

import numpy as np

import mindspore

import mindspore.dataset as ds

import mindspore.nn as nn

from mindspore import context

from mindspore.nn.metrics import Accuracy

from mindspore.train import Model

from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor

from mindspore import Tensor

context.set\_context(mode=context.GRAPH\_MODE, device\_target='Ascend')

* 1. 变量定义

cfg = edict({

    'train\_size': 60000,  # 训练集大小

    'test\_size': 10000,  # 测试集大小

    'channel': 1,  # 图片通道数

    'image\_height': 28,  # 图片高度

    'image\_width': 28,  # 图片宽度

    'batch\_size': 60,

    'num\_classes': 10,  # 分类类别

    'lr': 0.001,  # 学习率

    'epoch\_size': 20,  # 训练次数

    #此处应该改为自己数据集存储的路径，分别用train和test两个文件夹存储训练数据集和测试数据集

    'data\_dir\_train': os.path.join('./Fashion-MNIST/train/'),

    'data\_dir\_test': os.path.join('./Fashion-MNIST/test/'),

    'save\_checkpoint\_steps': 1,  # 多少步保存一次模型

    'keep\_checkpoint\_max': 3,  # 最多保存多少个模型

    'output\_directory': './model\_fashion',  # 保存模型路径

    'output\_prefix': "checkpoint\_fashion\_forward"  # 保存模型文件名字

})

* 1. 读取并处理数据

步骤 1读取数据

def read\_image(file\_name):

    '''

    :param file\_name: 文件路径

    :return:  训练或者测试数据

    如下是训练的图片的二进制格式

    [offset] [type]          [value]          [description]

    0000     32 bit integer  0x00000803(2051) magic number

    0004     32 bit integer  60000            number of images

    0008     32 bit integer  28               number of rows

    0012     32 bit integer  28               number of columns

    0016     unsigned byte   ??               pixel

    0017     unsigned byte   ??               pixel

    ........

    xxxx     unsigned byte   ??               pixel

    '''

    file\_handle = open(file\_name, "rb")  # 以二进制打开文档

    file\_content = file\_handle.read()  # 读取到缓冲区中

    head = struct.unpack\_from('>IIII', file\_content, 0)  # 取前4个整数，返回一个元组

    offset = struct.calcsize('>IIII')

    imgNum = head[1]  # 图片数

    width = head[2]  # 宽度

    height = head[3]  # 高度

    bits = imgNum \* width \* height  # data一共有60000\*28\*28个像素值

    bitsString = '>' + str(bits) + 'B'  # fmt格式：'>47040000B'

    imgs = struct.unpack\_from(bitsString, file\_content, offset)  # 取data数据，返回一个元组

    imgs\_array = np.array(imgs).reshape((imgNum, width \* height))  # 最后将读取的数据reshape成 【图片数，图片像素】二维数组

    return imgs\_array

def read\_label(file\_name):

    '''

    :param file\_name:

    :return:

    标签的格式如下：

    [offset] [type]          [value]          [description]

    0000     32 bit integer  0x00000801(2049) magic number (MSB first)

    0004     32 bit integer  60000            number of items

    0008     unsigned byte   ??               label

    0009     unsigned byte   ??               label

    ........

    xxxx     unsigned byte   ??               label

    The labels values are 0 to 9.

    '''

    file\_handle = open(file\_name, "rb")  # 以二进制打开文档

    file\_content = file\_handle.read()  # 读取到缓冲区中

    head = struct.unpack\_from('>II', file\_content, 0)  # 取前2个整数，返回一个元组

    offset = struct.calcsize('>II')

    labelNum = head[1]  # label数

    bitsString = '>' + str(labelNum) + 'B'  # fmt格式：'>47040000B'

    label = struct.unpack\_from(bitsString, file\_content, offset)  # 取data数据，返回一个元组

return np.array(label)

def get\_data():

    # 文件获取

    train\_image = os.path.join(cfg.data\_dir\_train, 'train-images-idx3-ubyte')

    test\_image = os.path.join(cfg.data\_dir\_test, "t10k-images-idx3-ubyte")

    train\_label = os.path.join(cfg.data\_dir\_train, "train-labels-idx1-ubyte")

    test\_label = os.path.join(cfg.data\_dir\_test, "t10k-labels-idx1-ubyte")

    # 读取数据

    train\_x = read\_image(train\_image)

    test\_x = read\_image(test\_image)

    train\_y = read\_label(train\_label)

    test\_y = read\_label(test\_label)

    return train\_x, train\_y, test\_x, test\_y

步骤 2数据预处理和处理结果图片展示

print(os.getcwd())



train\_x, train\_y, test\_x, test\_y = get\_data()

#第一维度是batchsize的数据,第二维度是图像的channel数，第三，第四维度是高和宽

train\_x = train\_x.reshape(-1, 1, cfg.image\_height, cfg.image\_width)

test\_x = test\_x.reshape(-1, 1, cfg.image\_height, cfg.image\_width)

#归一化至0和1之间

train\_x = train\_x / 255.0

test\_x = test\_x / 255.0

#修改数据格式

train\_x = train\_x.astype('float32')

test\_x = test\_x.astype('float32')

train\_y = train\_y.astype('int32')

test\_y = test\_y.astype('int32')

print('训练数据集样本数：', train\_x.shape[0])

print('测试数据集样本数：', test\_y.shape[0])

print('通道数/图像长/宽：', train\_x.shape[1:])

print('一张图像的标签样式：', train\_y[0])  # 一共10类，用0-9的数字表达类别。

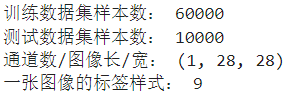
plt.figure()

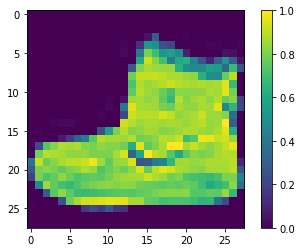
plt.imshow(train\_x[0,0,...])

plt.colorbar()

plt.grid(False)

plt.show()





步骤 3使用MindSpore GeneratorDataset接口将numpy.ndarray类型的数据转换为Dataset

# 转换数据类型为Dataset

XY\_train = list(zip(train\_x, train\_y))

#转换数据和标签为dataset类型，并制定数据为x，标签为y

ds\_train = ds.GeneratorDataset(XY\_train, ['x', 'y'])

ds\_train = ds\_train.shuffle(buffer\_size=cfg.train\_size).batch(cfg.batch\_size, drop\_remainder=True)

XY\_test = list(zip(test\_x, test\_y))

ds\_test = ds.GeneratorDataset(XY\_test, ['x', 'y'])

ds\_test = ds\_test.shuffle(buffer\_size=cfg.test\_size).batch(cfg.batch\_size, drop\_remainder=True)

* 1. 定义前馈神经网络

前馈神经网络是一种最简单的神经网络，各神经元分层排列（其中每一层包含若干个神经元）。每个神经元只与前一层的神经元相连，接收前一层的输出，并输出给下一层，各层间没有反馈。是目前应用最广泛、发展最迅速的人工神经网络之一。第0层叫输入层，最后一层叫输出层，其他中间层叫做隐含层（或隐藏层、隐层）。隐层可以是一层，也可以是多层，是由全连接层堆叠而成。

# 定义前馈神经网络

class Forward\_fashion(nn.Cell):

    def \_\_init\_\_(self, num\_class=10):  # 一共分十类，图片通道数是1

        super(Forward\_fashion, self).\_\_init\_\_()

        self.num\_class = num\_class

        self.flatten = nn.Flatten()

        self.fc1 = nn.Dense(cfg.channel \* cfg.image\_height \* cfg.image\_width, 128)

        self.relu = nn.ReLU()

        self.fc2 = nn.Dense(128, self.num\_class)

    def construct(self, x):

        x = self.flatten(x)

        x = self.fc1(x)

        x = self.relu(x)

        x = self.fc2(x)

        return x

* 1. 训练

使用Fashion-MNIST数据集对上述定义的前馈神经网络模型进行训练。训练策略如下表所示，可以调整训练策略并查看训练效果。

# 构建网络

network = Forward\_fashion(cfg.num\_classes)

# 定义模型的损失函数，优化器

net\_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")

net\_opt = nn.Adam(network.trainable\_params(), cfg.lr)

# 训练模型

model = Model(network, loss\_fn=net\_loss, optimizer=net\_opt, metrics={"acc"})

loss\_cb = LossMonitor(per\_print\_times=int(cfg.train\_size / cfg.batch\_size))

#设定每多少step保存一个checkpoint，并且设定最多保存多少个checkpoint

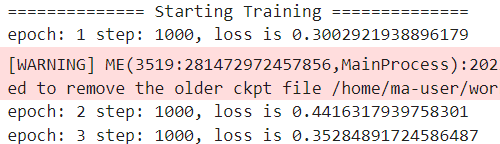
config\_ck = CheckpointConfig(save\_checkpoint\_steps=cfg.save\_checkpoint\_steps,

                             keep\_checkpoint\_max=cfg.keep\_checkpoint\_max)

ckpoint\_cb = ModelCheckpoint(prefix=cfg.output\_prefix, directory=cfg.output\_directory, config=config\_ck)

print("============== Starting Training ==============")

model.train(cfg.epoch\_size, ds\_train, callbacks=[ckpoint\_cb, loss\_cb], dataset\_sink\_mode=False)



* 1. 评估测试

# 使用测试集评估模型，打印总体准确率

metric = model.eval(ds\_test)

print(metric)



class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

#从测试集中取出一组样本，输入模型进行预测

test\_ = ds\_test.create\_dict\_iterator().\_\_next\_\_()

#利用key值选出样本

test = Tensor(test\_['x'], mindspore.float32)

predictions = model.predict(test)

softmax = nn.Softmax()

predictions = softmax(predictions)

predictions = predictions.asnumpy()

true\_label = test\_['y'].asnumpy()

for i in range(15):

    p\_np = predictions[i, :]

    pre\_label = np.argmax(p\_np)

    print('第' + str(i) + '个sample预测结果：', class\_names[pre\_label], '   真实结果：', class\_names[true\_label[i]])



* 1. 对预测结果可视化

# -------------------定义可视化函数--------------------------------

# 输入预测结果序列，真实标签序列，以及图片序列

# 目标是根据预测值对错，让其标签显示为红色或者蓝色。对：标签为蓝色；错：标签为红色

def plot\_image(predicted\_label, true\_label, img):

    plt.grid(False)

    plt.xticks([])

    plt.yticks([])

    # 显示对应图片

    plt.imshow(img, cmap=plt.cm.binary)

    # 显示预测结果的颜色，如果对上了是蓝色，否则为红色

    if predicted\_label == true\_label:

        color = 'blue'

    else:

        color = 'red'

    # 显示对应标签的格式，样式

    plt.xlabel('{},({})'.format(class\_names[predicted\_label],

                                    class\_names[true\_label]), color=color)

# 将预测的结果以柱状图形状显示蓝对红错

def plot\_value\_array(predicted\_label, true\_label,predicted\_array):

    plt.grid(False)

    plt.xticks([])

    plt.yticks([])

    this\_plot = plt.bar(range(10), predicted\_array, color='#777777')

    plt.ylim([0, 1])

    this\_plot[predicted\_label].set\_color('red')

    this\_plot[true\_label].set\_color('blue')

# 预测15个图像与标签，并展现出来

num\_rows = 5

num\_cols = 3

num\_images = num\_rows \* num\_cols

plt.figure(figsize=(2 \* 2 \* num\_cols, 2 \* num\_rows))

for i in range(num\_images):

    plt.subplot(num\_rows, 2 \* num\_cols, 2 \* i + 1)

    pred\_np\_ = predictions[i, :]

    predicted\_label = np.argmax(pred\_np\_)

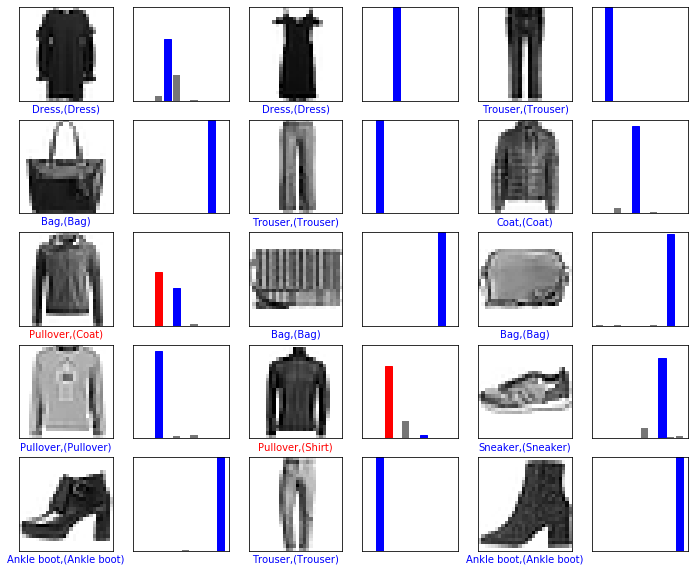
    image\_single = test\_['x'][i, 0, ...].asnumpy()

    plot\_image(predicted\_label, true\_label[i], image\_single)

    plt.subplot(num\_rows, 2 \* num\_cols, 2 \* i + 2)

    plot\_value\_array(predicted\_label, true\_label[i], pred\_np\_)

plt.show()



1. 鸢尾花分类任务对比实验
   1. 构建Beale 公式及其导数

步骤 1 导入相关模块

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.colors as plt\_cl  # Matplotlib的色阶条

步骤 2 构建相关函数

#定义beale公式

def beale(x1,x2):

    return (1.5-x1+x1\*x2)\*\*2+(2.25-x1+x1\*x2\*\*2)\*\*2+(2.625-x1+x1\*x2\*\*3)\*\*2

#定义beale公式的偏导函数

def dbeale\_dx(x1, x2):

    dfdx1 = 2\*(1.5-x1+x1\*x2)\*(x2-1)+2\*(2.25-x1+x1\*x2\*\*2)\*(x2\*\*2-1)+2\*(2.625-x1+x1\*x2\*\*3)\*(x2\*\*3-1) # 求beale公式关于x1的偏导数

    dfdx2 = 2\*(1.5-x1+x1\*x2)\*x1+2\*(2.25-x1+x1\*x2\*\*2)\*(2\*x1\*x2)+2\*(2.625-x1+x1\*x2\*\*3)\*(3\*x1\*x2\*\*2) # 求beale公式关于x2的偏导数

    return dfdx1, dfdx2

步骤 3 可视化beale方程

# 定义画图函数

def gd\_plot(x\_traj):

    plt.rcParams['figure.figsize'] = [6, 6] # 窗口大小

    plt.contour(X1, X2, Y, levels=np.logspace(0, 6, 30),

                norm=plt\_cl.LogNorm(), cmap=plt.cm.jet)  # 画等高线图

    plt.title('2D Contour Plot of Beale function(Momentum)') # 添加标题

    plt.xlabel('$x\_1$') # x轴标签

    plt.ylabel('$x\_2$') # y轴标签

    plt.axis('equal') # 设置坐标轴为正方形

    plt.plot(3, 0.5, 'k\*', markersize=10) # 画出最低点

    if x\_traj is not None:

        x\_traj = np.array(x\_traj) # 将x\_traj转为数组

        plt.plot(x\_traj[:, 0], x\_traj[:, 1], 'k-')

# 以x\_traj的第一列为x轴坐标，第二列为y轴坐标进行画图

plt.show() # 显示图像

step\_x1, step\_x2 = 0.2, 0.2

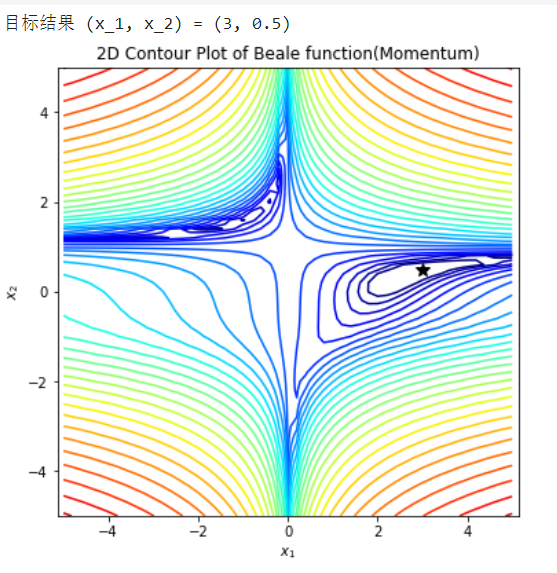
X1, X2 = np.meshgrid(np.arange(-5, 5 + step\_x1, step\_x1),

                     np.arange(-5, 5 + step\_x2, step\_x2))  # 将图形从-5 到 5.2，步长为0.2 划分成网格点

Y = beale(X1, X2) # 将x1,x2坐标带入beale公式

print("目标结果 (x\_1, x\_2) = (3, 0.5)")

gd\_plot(None) # 调用函数



* 1. 编写不使用优化器优化Beale 公式

#定义无优化器函数

def gd\_no(df\_dx, x0, conf\_para=None):

    if conf\_para is None:

        conf\_para = {}

    conf\_para.setdefault('n\_iter', 1000)  # 迭代次数

    conf\_para.setdefault('learning\_rate', 0.001)  # 设置学习率

    x\_traj = []

    x\_traj.append(x0)

    v = np.zeros\_like(x0)

#没有迭代更新的操作，所以，坐标没有变化

    for iter in range(1, conf\_para['n\_iter'] + 1):

        x\_traj.append(x\_traj[-1])

    return x\_traj

#初始化坐标

x0 = np.array([1.0, 1.5])

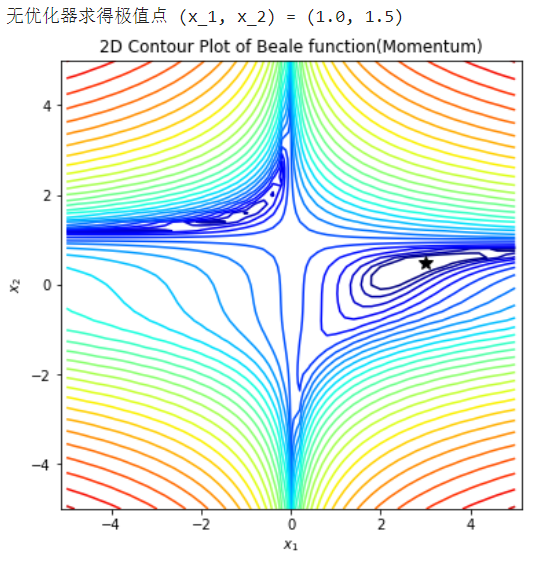
conf\_para\_no = {'n\_iter': 2000, 'learning\_rate': 0.005}

#调用函数进行更新

x\_traj\_no = gd\_no(dbeale\_dx, x0, conf\_para\_no)

print("无优化器求得极值点 (x\_1, x\_2) = (%s, %s)" % (x\_traj\_no[-1][0], x\_traj\_no[-1][1]))

gd\_plot(x\_traj\_no)



* 1. 编写SGD优化器并优化Beale 公式

def gd\_sgd(df\_dx, x0, conf\_para=None):

    if conf\_para is None:

        conf\_para = {}

    conf\_para.setdefault('n\_iter', 1000)  # 迭代次数

    conf\_para.setdefault('learning\_rate', 0.001)  # 设置学习率

    x\_traj = []

    x\_traj.append(x0)

    v = np.zeros\_like(x0)

#利用梯度值对坐标进行更新

    for iter in range(1, conf\_para['n\_iter'] + 1):

        dfdx = np.array(df\_dx(x\_traj[-1][0], x\_traj[-1][1]))  # 计算梯度

        v = - conf\_para['learning\_rate'] \* dfdx  # 负的学习率\*梯度值

        x\_traj.append(x\_traj[-1] + v)  # 添加新的x\_traj

    return x\_traj

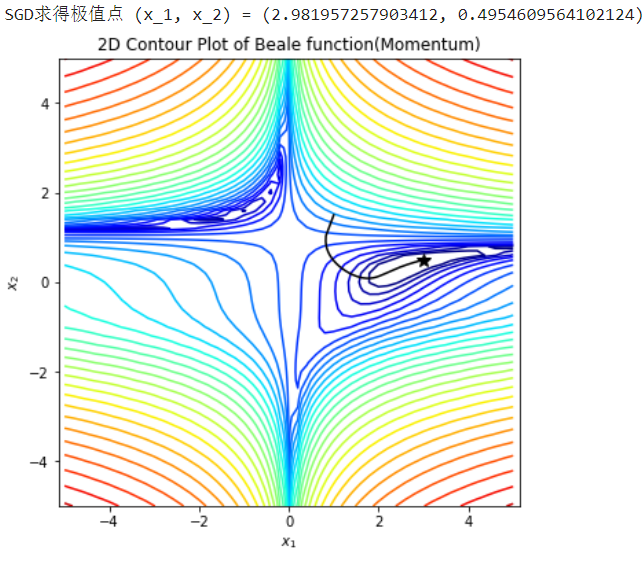
x0 = np.array([1.0, 1.5])

conf\_para\_sgd = {'n\_iter': 2000, 'learning\_rate': 0.005}

x\_traj\_sgd = gd\_sgd(dbeale\_dx, x0, conf\_para\_sgd)

print("SGD求得极值点 (x\_1, x\_2) = (%s, %s)" % (x\_traj\_sgd[-1][0], x\_traj\_sgd[-1][1]))

gd\_plot(x\_traj\_sgd)



* 1. 编写动量优化器并优化Beale 公式

def gd\_momentum(df\_dx, x0, conf\_para=None):

    if conf\_para is None:

        conf\_para = {}

    conf\_para.setdefault('n\_iter', 1000)  # 迭代次数

    conf\_para.setdefault('learning\_rate', 0.001)  # 设置学习率

    conf\_para.setdefault('momentum', 0.9)  # 设置动量参数

    x\_traj = []

    x\_traj.append(x0)

    v = np.zeros\_like(x0)

#套用动量优化器公式，对坐标值进行更新

    for iter in range(1, conf\_para['n\_iter'] + 1):

        dfdx = np.array(df\_dx(x\_traj[-1][0], x\_traj[-1][1]))  # 计算梯度

        v = conf\_para['momentum'] \* v - conf\_para['learning\_rate'] \* dfdx  # 计算更新

        x\_traj.append(x\_traj[-1] + v)

    return x\_traj

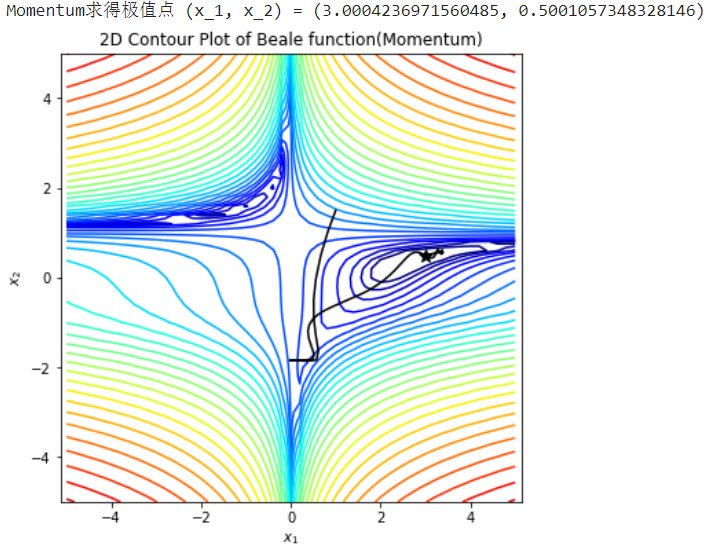
x0 = np.array([1.0, 1.5])

conf\_para\_momentum = {'n\_iter': 500, 'learning\_rate': 0.005}

x\_traj\_momentum = gd\_momentum(dbeale\_dx, x0, conf\_para\_momentum)

print("Momentum求得极值点 (x\_1, x\_2) = (%s, %s)" % (x\_traj\_momentum[-1][0], x\_traj\_momentum[-1][1]))

gd\_plot(x\_traj\_momentum)



* 1. 编写自适应优化器并优化Beale 公式

def gd\_adagrad(df\_dx, x0, conf\_para=None):

    if conf\_para is None:

        conf\_para = {}

    conf\_para.setdefault('n\_iter', 1000)  # 迭代次数

    conf\_para.setdefault('learning\_rate', 0.001)  # 学习率

    conf\_para.setdefault('epsilon', 1e-7)

    x\_traj = []

    x\_traj.append(x0)

    r = np.zeros\_like(x0)

   #套用adagrad优化器公式，对参数进行更新

    for iter in range(1, conf\_para['n\_iter'] + 1):

        dfdx = np.array(df\_dx(x\_traj[-1][0], x\_traj[-1][1]))

        r += dfdx \*\* 2

        x\_traj.append(x\_traj[-1] - conf\_para['learning\_rate'] / (np.sqrt(r) + conf\_para['epsilon']) \* dfdx)

    return x\_traj

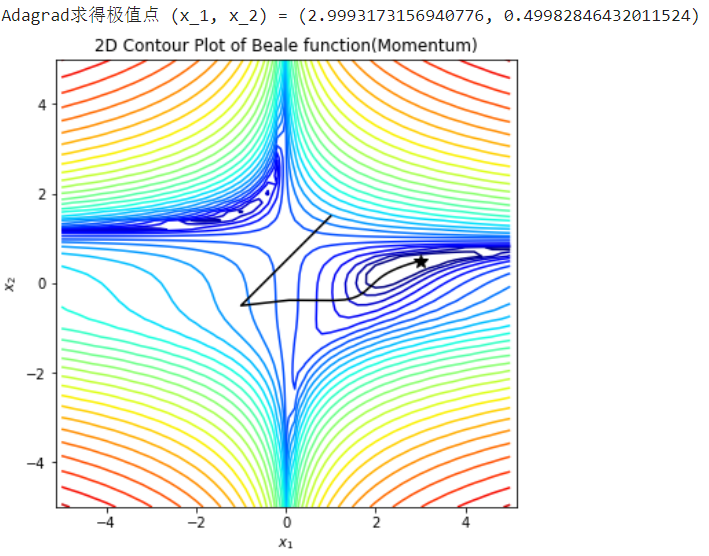
x0 = np.array([1.0, 1.5])

conf\_para\_adag = {'n\_iter': 500, 'learning\_rate': 2}

x\_traj\_adag = gd\_adagrad(dbeale\_dx, x0, conf\_para\_adag)

print("Adagrad求得极值点 (x\_1, x\_2) = (%s, %s)" % (x\_traj\_adag[-1][0], x\_traj\_adag[-1][1]))

gd\_plot(x\_traj\_adag)



* 1. 导入模块、读取鸢尾花数据集并预处理、构建神经网络

步骤 1导入模块和定义变量

import csv

import os

import time

import numpy as np

from easydict import EasyDict as edict

from matplotlib import pyplot as plt

import mindspore

from mindspore import nn

from mindspore import context

from mindspore import dataset

from mindspore.train.callback import TimeMonitor, LossMonitor

from mindspore import Tensor

from mindspore.train import Model

from mindspore.train.callback import ModelCheckpoint, CheckpointConfig

context.set\_context(mode=context.GRAPH\_MODE, device\_target="Ascend")  # 设定运行模式为静态图模式，并且运行设备为昇腾芯片

步骤 2 读取数据并预处理

#变量定义

cfg = edict({

    'data\_size': 150,

    'train\_size': 120,      #训练集大小

    'test\_size': 30 ,       #测试集大小

    'feature\_number': 4,       #输入特征数

    'num\_class': 3,     #分类类别

    'batch\_size': 30,   #批次大小

    'data\_dir':    'iris.data',     # 数据集路径

    'save\_checkpoint\_steps': 5,                 #多少步保存一次模型

    'keep\_checkpoint\_max': 1,                      #最多保存多少个模型

    'out\_dir\_no\_opt':   './model\_iris/no\_opt',          #保存模型路径，无优化器模型

    'out\_dir\_sgd':   './model\_iris/sgd',          #保存模型路径,SGD优化器模型

    'out\_dir\_momentum':   './model\_iris/momentum',          #保存模型路径，momentum模型

    'out\_dir\_adam':   './model\_iris/adam',          #保存模型路径，adam优化器模型

    'output\_prefix': "checkpoint\_fashion\_forward"     #保存模型文件名

})

步骤 3定义训练、评估、测试函数

#鸢尾花数据集，本数据集共有150个带标签的数据

with open(cfg.data\_dir) as csv\_file:

    data = list(csv.reader(csv\_file, delimiter=','))

label\_map = {'setosa': 0,'versicolor': 1,'virginica':2 }

#分别获取数据中的特征值X和标签值Y

X = np.array([[float(x) for x in s[:-1]] for s in data[:cfg.data\_size]], np.float32)

Y = np.array([label\_map[s[-1]] for s in data[:cfg.data\_size]], np.int32)

# 将数据集分为训练集120条，测试集30条。

train\_idx = np.random.choice(cfg.data\_size, cfg.train\_size, replace=False)

test\_idx = np.array(list(set(range(cfg.data\_size)) - set(train\_idx)))

X\_train, Y\_train = X[train\_idx], Y[train\_idx]

X\_test, Y\_test = X[test\_idx], Y[test\_idx]

def gen\_data(X\_train, Y\_train, epoch\_size):

    #生成训练集

    XY\_train = list(zip(X\_train, Y\_train))

    ds\_train = dataset.GeneratorDataset(XY\_train, ['x', 'y'])  # x, y 为列名

    #设定数据集大小

    #打乱操作并设定batchsize

    ds\_train = ds\_train.shuffle(buffer\_size=cfg.train\_size).batch(cfg.batch\_size, drop\_remainder=True) # buffer\_size：数据集大小，drop\_remainder：舍弃最后一个不完整的batch

    #生成测试集

    XY\_test = list(zip(X\_test, Y\_test))

    ds\_test = dataset.GeneratorDataset(XY\_test, ['x', 'y'])

    #设定数据集大小

    #打乱操作并设定batchsize

    ds\_test = ds\_test.shuffle(buffer\_size=cfg.test\_size).batch(cfg.test\_size, drop\_remainder=True)

return ds\_train, ds\_test

# 训练函数

def train(network, net\_opt, ds\_train, prefix, directory, print\_times):

    #定义网络损失函数

    net\_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")  # sparse为True时对Label数据做one\_hot处理，reduction支持mean和sum

    #定义模型

    model = Model(network, loss\_fn=net\_loss, optimizer=net\_opt, metrics={"acc"}) # 定义网络结构，损失函数，优化器，评估方式

    #定义损失值指标

    loss\_cb = LossMonitor(per\_print\_times=print\_times)   # 每隔 print\_times 步监测一下损失值

#     loss\_cb = LossMonitor(per\_print\_times=int(cfg.train\_size / cfg.batch\_size))   # 每隔4步监测一下损失值

    #设置checkpoint

    config\_ck = CheckpointConfig(save\_checkpoint\_steps=cfg.save\_checkpoint\_steps,    # 每5步保存一下模型

                                 keep\_checkpoint\_max=cfg.keep\_checkpoint\_max)        # 最多保存1个模型

    ckpoint\_cb = ModelCheckpoint(prefix=prefix, directory=directory, config=config\_ck)  # 设置文件名，文件路径，以及checkpoint参数

    print("============== Starting Training ==============")

    #训练模型

    model.train(epoch\_size, ds\_train, callbacks=[ckpoint\_cb, loss\_cb], dataset\_sink\_mode=False) # 设置训练次数，训练数据，回调函数（checkpoint和lossmonitor），Ascend是否采用下沉模式

return model

class\_names=['setosa', 'versicolor', 'virginica']

# 评估预测函数

def eval\_predict(model, ds\_test):

    # 使用测试集评估模型，打印总体准确率

    metric = model.eval(ds\_test)

    print(metric)

    # 预测

    test\_ = ds\_test.create\_dict\_iterator().\_\_next\_\_()  # 生成测试集

    test = Tensor(test\_['x'], mindspore.float32)  # 将测试集的特征转换成mindspore数据类型

    predictions = model.predict(test)  # 用predict进行预测

    predictions = predictions.asnumpy()  # 将预测值转换成numpy数组类型, predictions.shape为(30, 3)

    true\_label = test\_['y'].asnumpy()  # 将真实值转换成numpy数组类型

    for i in range(10):

        p\_np = predictions[i, :]  # 取第i个数据的预测值

        pre\_label = np.argmax(p\_np)  # 取最大值的索引作为输出标签

        print('第' + str(i) + '个sample预测结果：', class\_names[pre\_label], '   真实结果：', class\_names[true\_label[i]])  # 输出预测值和真实值的对比结果

* 1. 无优化器模型训练并预测

epoch\_size = 20   # 训练20次

print('------------------无优化器--------------------------')

# 数据

ds\_train, ds\_test = gen\_data(X\_train, Y\_train, epoch\_size)  # 生成训练集和测试集

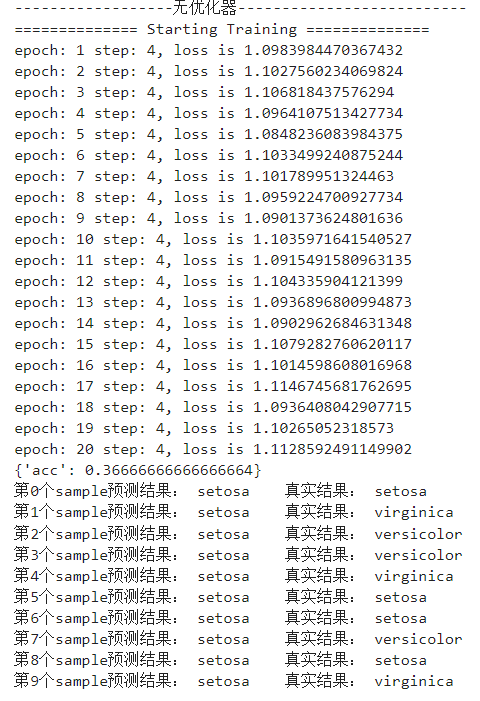
# 定义网络并训练

network = nn.Dense(cfg.feature\_number, cfg.num\_class)  # 定义一个全连接网络层，输入特征为4，输出类别为3

model = train(network, None, ds\_train, "checkpoint\_no\_opt", cfg.out\_dir\_no\_opt, print\_times=4)  # 用训练集训练网络，设置网络结构，模型名称，保存路径, print\_times

# 评估预测

eval\_predict(model, ds\_test)  # 用测试集进行预测



* 1. SGD优化器模型训练并预测

epoch\_size = 20   # 训练20次

print('------------------无优化器--------------------------')

# 数据

ds\_train, ds\_test = gen\_data(X\_train, Y\_train, epoch\_size)  # 生成训练集和测试集

# 定义网络并训练

network = nn.Dense(cfg.feature\_number, cfg.num\_class)  # 定义一个全连接网络层，输入特征为4，输出类别为3

model = train(network, None, ds\_train, "checkpoint\_no\_opt", cfg.out\_dir\_no\_opt, print\_times=4)  # 用训练集训练网络，设置网络结构，模型名称，保存路径, print\_times

# 评估预测

eval\_predict(model, ds\_test)  # 用测试集进行预测

epoch\_size = 300  # 训练300次

lr = 0.01

print('-------------------SGD优化器-----------------------')

# 数据

ds\_train, ds\_test = gen\_data(X\_train, Y\_train, epoch\_size) # 生成训练集和测试集

# 定义网络并训练、测试、预测

network = nn.Dense(cfg.feature\_number, cfg.num\_class)  # 定义一个全连接网络层，输入特征为4，输出类别为3

net\_opt = nn.SGD(network.trainable\_params(), lr)  # 用SGD优化器进行优化

model = train(network, net\_opt, ds\_train, "checkpoint\_sgd", cfg.out\_dir\_sgd, 40)  # 用训练集训练网络，设置网络结构，优化器，模型名称，保存路径, print\_times

# 评估预测

eval\_predict(model, ds\_test) # 用测试集进行预测

-------------------SGD优化器-----------------------

============== Starting Training ==============

epoch: 10 step: 4, loss is 0.8733811974525452

epoch: 20 step: 4, loss is 0.7710245251655579

epoch: 30 step: 4, loss is 0.7947858572006226

epoch: 40 step: 4, loss is 0.5786421298980713

epoch: 50 step: 4, loss is 0.6049482226371765

epoch: 60 step: 4, loss is 0.4949568510055542

epoch: 70 step: 4, loss is 0.5240433812141418

epoch: 80 step: 4, loss is 0.5319739580154419

epoch: 90 step: 4, loss is 0.5064218044281006

epoch: 100 step: 4, loss is 0.4477822184562683

epoch: 110 step: 4, loss is 0.4583004415035248

epoch: 120 step: 4, loss is 0.45691412687301636

epoch: 130 step: 4, loss is 0.48073580861091614

epoch: 140 step: 4, loss is 0.42902177572250366

epoch: 150 step: 4, loss is 0.4827762246131897

epoch: 160 step: 4, loss is 0.39740049839019775

epoch: 170 step: 4, loss is 0.4294254779815674

epoch: 180 step: 4, loss is 0.458183228969574

epoch: 190 step: 4, loss is 0.34379738569259644

epoch: 200 step: 4, loss is 0.3158137798309326

epoch: 210 step: 4, loss is 0.3819582462310791

epoch: 220 step: 4, loss is 0.39689674973487854

epoch: 230 step: 4, loss is 0.42388272285461426

epoch: 240 step: 4, loss is 0.2960008978843689

epoch: 250 step: 4, loss is 0.37860193848609924

epoch: 260 step: 4, loss is 0.38026177883148193

epoch: 270 step: 4, loss is 0.32997390627861023

epoch: 280 step: 4, loss is 0.3754065930843353

epoch: 290 step: 4, loss is 0.3316059112548828

epoch: 300 step: 4, loss is 0.3462972044944763

{'acc': 0.9}

第0个sample预测结果： setosa    真实结果： setosa

第1个sample预测结果： virginica    真实结果： versicolor

第2个sample预测结果： virginica    真实结果： virginica

第3个sample预测结果： setosa    真实结果： setosa

第4个sample预测结果： virginica    真实结果： virginica

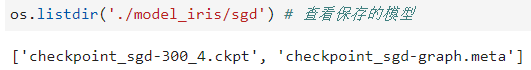
第5个sample预测结果： versicolor    真实结果： versicolor

第6个sample预测结果： versicolor    真实结果： versicolor

第7个sample预测结果： virginica    真实结果： virginica

第8个sample预测结果： setosa    真实结果： setosa

第9个sample预测结果： versicolor    真实结果： versicolor



* 1. Momentum优化器模型训练并预测

epoch\_size = 20  # 训练20次

lr = 0.01  # 学习率为0.01

print('-------------------Momentum优化器-----------------------')

# 数据

ds\_train, ds\_test = gen\_data(X\_train, Y\_train, epoch\_size)  # 生成训练集和测试集

# 定义网络并训练

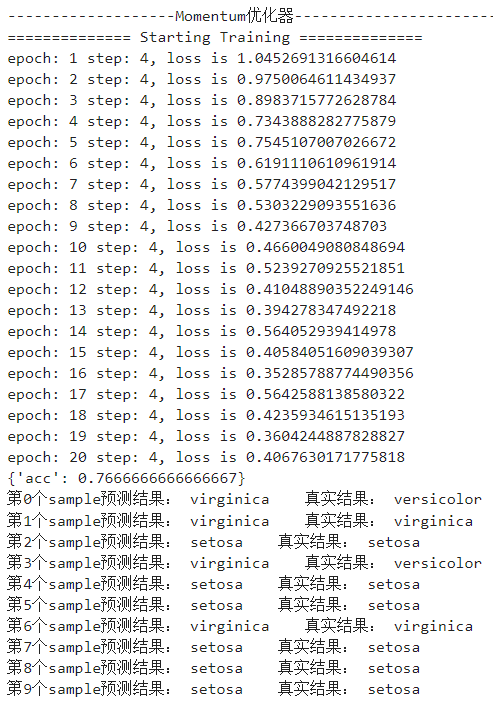
network = nn.Dense(cfg.feature\_number, cfg.num\_class) # 定义一个全连接网络层，输入特征为4，输出类别为3

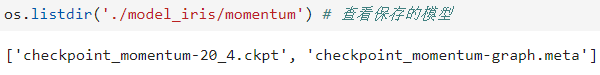
net\_opt = nn.Momentum(network.trainable\_params(), lr, 0.9)   # 用 momentum 优化器进行优化，学习率为0.01，动量大小为0.9

model = train(network, net\_opt, ds\_train, "checkpoint\_momentum", cfg.out\_dir\_momentum, 4)  # 用训练集训练网络，设置网络结构，优化器，模型名称，保存路径, print\_times

# 评估预测

eval\_predict(model, ds\_test)  # 用测试集进行预测





* 1. Adam优化器模型训练并预测

epoch\_size = 15  # 训练15次

lr = 0.1  # 学习率为0.1, 动态学习率

print('------------------Adam优化器--------------------------')

# 数据

ds\_train, ds\_test = gen\_data(X\_train, Y\_train, epoch\_size)  # 生成训练集和测试集

# 定义网络并训练

network = nn.Dense(cfg.feature\_number, cfg.num\_class)  # 定义一个全连接网络层，输入特征为4，输出类别为3

net\_opt = nn.Adam(network.trainable\_params(), learning\_rate=lr)  # 用 Adam 优化器进行优化，学习率为0.1

model = train(network, net\_opt, ds\_train, "checkpoint\_adam", cfg.out\_dir\_adam, 4)  # 用训练集训练网络，设置网络结构，优化器，模型名称，保存路径, print\_times

# 评估预测

eval\_predict(model, ds\_test)



1. 实验结果

1. 无优化器训练loss基本没有发生变化，测试结果效果差。多运行几次发现结果偏差太大。

2. SGD优化器loss下降速度很慢，而且在接近收敛处loss下降非常缓慢。增大学习率，减少迭代次数，会出现收敛到局部最优解的情况。

3. Momentum优化器loss下降速度较快，充分说明Momentum优化器改进了SGD收敛速度慢的问题。改变参数，比较不同学习率和迭代次数的结果，会发现该优化器稳定性很强，学习率容易选择。相比于SGD优化器容易调参。

4. adam优化器loss下降速度最快快，只需要15 epoch就可以达到收敛。改变模型学习率多跑几次，会发现，adam优化器可以适应不用的学习率，容易调节参数。

1. 实验总结

本实验目的是使用MindSpore对比不同优化器算法，通过比较不同优化器实验结果，得到如下结论：

（1）无优化器条件训练，不更新参数，训练loss基本不变。模型与初始化参数关系大，随机性较强，效果差，不适合应用；

（2）有优化器条件下训练，更新模型参数。训练loss改变；

（3）SGD优化器收敛速度慢，参数不容易调节；

（4）Momentum优化器是SGD加入历史梯度动量，增加了稳定性，收敛速度有明显提高，参数较SGD容易调节；

（5）Adam优化器是 RMSprop + Momentum。可以适应不同学习率，可以自动为不同参数调节学习率。稳定性高，参数容易调节；