深度学习基础

实验二

版本：2.2



华为技术有限公司

|  |
| --- |
| **版权所有 © 华为技术有限公司 2021。 保留一切权利。**  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  **商标声明**  C:\Users\jwx341670\Desktop\华为标志 Huawei Logo 2018\竖版标志Vertical Version\PNG\HW_POS_RBG_Vertical-150ppi.png 和其他华为商标均为华为技术有限公司的商标。  本文档提及的其他所有商标或注册商标，由各自的所有人拥有。  **注意**  您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | http://[e](http://e.huawei.com/).huawei.com |

目录

[1 实验介绍 6](#_Toc92877225)

[1.1 实验目的 6](#_Toc92877226)

[1.2 实验清单 6](#_Toc92877227)

[1.3 实验开发环境 7](#_Toc92877228)

[1.4 开发平台介绍 7](#_Toc92877229)

[1.5 背景知识 7](#_Toc92877230)

[2 手写体图像识别实验 10](#_Toc92877231)

[2.1 实验介绍 10](#_Toc92877232)

[2.1.1 简介 10](#_Toc92877233)

[2.1.2 实验目的 10](#_Toc92877234)

[2.2 实验环境要求 10](#_Toc92877235)

[2.3 实验总体设计 10](#_Toc92877236)

[2.4 实验过程 11](#_Toc92877237)

[2.4.1 创建实验环境 11](#_Toc92877238)

[2.4.2 导入实验所需模块 12](#_Toc92877239)

[2.4.3 导入实验数据集 12](#_Toc92877240)

[2.4.4 模型搭建与训练 14](#_Toc92877241)

[2.4.5 模型评估 16](#_Toc92877242)

[2.4.6 停止实验环境 16](#_Toc92877243)

[2.5 实验总结 17](#_Toc92877244)

[2.6 创新设计 17](#_Toc92877245)

[3 FashionMnist图像分类实验 18](#_Toc92877246)

[3.1 实验介绍 18](#_Toc92877247)

[3.1.1 简介 18](#_Toc92877248)

[3.1.2 实验目的 18](#_Toc92877249)

[3.2 实验环境要求 18](#_Toc92877250)

[3.3 实验总体设计 18](#_Toc92877251)

[3.4 实验过程 19](#_Toc92877252)

[3.4.1 数据集准备 19](#_Toc92877253)

[3.4.2 创建实验环境 19](#_Toc92877254)

[3.4.3 导入实验所需模块 20](#_Toc92877255)

[3.4.4 变量定义 20](#_Toc92877256)

[3.4.5 读取并处理数据 21](#_Toc92877257)

[3.4.6 定义前馈神经网络 24](#_Toc92877258)

[3.4.7 训练 24](#_Toc92877259)

[3.4.8 评估测试 25](#_Toc92877260)

[3.4.9 对预测结果可视化 26](#_Toc92877261)

[3.4.10 停止实验环境 28](#_Toc92877262)

[3.5 实验总结 28](#_Toc92877263)

[3.6 创新设计 28](#_Toc92877264)

[4 汽车里程数回归预测实验 29](#_Toc92877265)

[4.1 实验介绍 29](#_Toc92877266)

[4.1.1 简介 29](#_Toc92877267)

[4.1.2 实验目的 29](#_Toc92877268)

[4.2 实验环境要求 29](#_Toc92877269)

[4.3 实验总体设计 29](#_Toc92877270)

[4.4 实验过程 30](#_Toc92877271)

[4.4.1 导入实验所需模块 30](#_Toc92877272)

[4.4.2 导入数据集并预处理 30](#_Toc92877273)

[4.4.3 模型搭建与训练 33](#_Toc92877274)

[4.4.4 查看模型训练状态 35](#_Toc92877275)

[4.4.5 停止实验环境 36](#_Toc92877276)

[4.5 实验总结 37](#_Toc92877277)

[4.6 创新设计 37](#_Toc92877278)

[5 鸢尾花分类任务对比实验 38](#_Toc92877279)

[5.1 实验介绍 38](#_Toc92877280)

[5.1.1 简介 38](#_Toc92877281)

[5.1.2 实验目的 38](#_Toc92877282)

[5.2 实验环境要求 38](#_Toc92877283)

[5.3 实验总体设计 38](#_Toc92877284)

[5.4 背景知识 39](#_Toc92877285)

[5.5 实验过程 40](#_Toc92877286)

[5.5.1 构建Beale 公式及其导数 40](#_Toc92877287)

[5.5.2 编写不使用优化器优化Beale 公式 41](#_Toc92877288)

[5.5.3 编写SGD优化器并优化Beale 公式 42](#_Toc92877289)

[5.5.4 编写动量优化器并优化Beale 公式 44](#_Toc92877290)

[5.5.5 编写自适应优化器并优化Beale 公式 45](#_Toc92877291)

[5.5.6 思考题 47](#_Toc92877292)

[5.5.7 导入模块、读取鸢尾花数据集并预处理、构建神经网络 47](#_Toc92877293)

[5.5.8 无优化器模型训练并预测 50](#_Toc92877294)

[5.5.9 SGD优化器模型训练并预测 51](#_Toc92877295)

[5.5.10 Momentum优化器模型训练并预测 52](#_Toc92877296)

[5.5.11 Adam优化器模型训练并预测 53](#_Toc92877297)

[5.5.12 查看保存的模型 54](#_Toc92877298)

[5.5.13 停止实验环境 54](#_Toc92877299)

[5.6 结果分析 54](#_Toc92877300)

[5.7 实验总结 55](#_Toc92877301)

[5.8 创新设计 55](#_Toc92877302)

[6 附录：ModelArts开发环境搭建 56](#_Toc92877303)

# 实验介绍

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。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。

本章实验主要专注于理解全连接网络（Fully Connected Neural Network）原理，并利用网络实现分类与回归任务。同时，对网络中应用到的优化器进行深入学习。

本章实验主要分为两个难度：初级和中级。

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。

中级实验：①鸢尾花分类任务对比实验。

## 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。

## 实验清单

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **实验** | **简述** | **难度** | **软件环境** | **开发环境** |
| 手写体图像识别实验 | 基于MindSpore框架，在ModelArts平台进行Mnist手写体图像识别实验。 | 初级 | MindSpore-1.7-python3.7 | ModelArts |
| FashionMnist图像分类实验 | 基于MindSpore框架，在本地进行FashionMnist图像分类实验。 | 初级 | MindSpore-1.7-python3.7 | ModelArts |
| 汽车里程数回归预测实验 | 基于MindSpore框架，在ModelArts进行汽车里程回归预测实验。 | 初级 | MindSpore-1.7-python3.7 | ModelArts |
| 鸢尾花分类任务对比实验 | 基于MindSpore框架，在ModelArts平台学习定义优化器并进行鸢尾花分类任务有无优化器对比实验。 | 中级 | MindSpore-1.7-python3.7 | ModelArts |

## 实验开发环境

Mindspore

若选择在华为云ModelArts上快速搭建开发环境，可参考文末附录：ModelArts开发环境搭建。

## 开发平台介绍

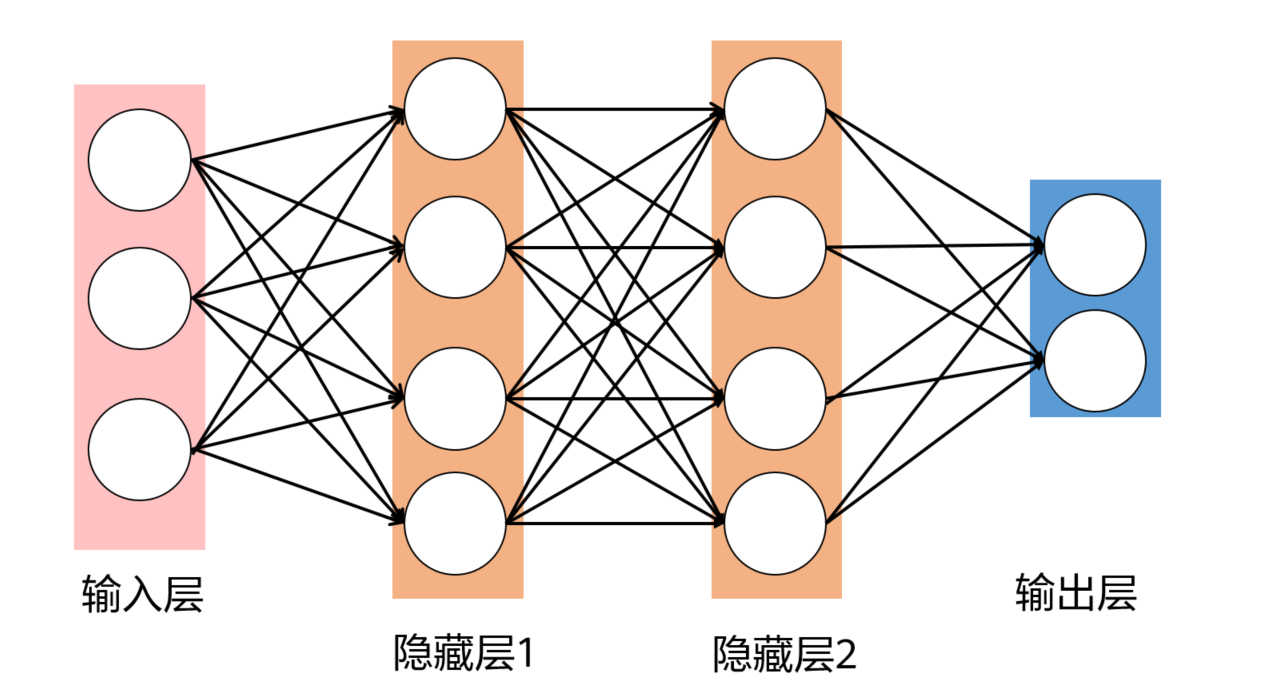
MindSpore 最佳匹配昇腾芯片的开源AI计算框架，支持Asend、GPU、CPU平台。MindSpore官网：<https://www.mindspore.cn>

ModelArts是面向开发者的一站式AI开发平台，为机器学习与深度学习提供海量数据预处理及半自动化标注、大规模分布式Training、自动化模型生成，及端-边-云模型按需部署能力，帮助用户快速创建和部署模型，管理全周期AI工作流。

具体内容请参考平台介绍ppt。

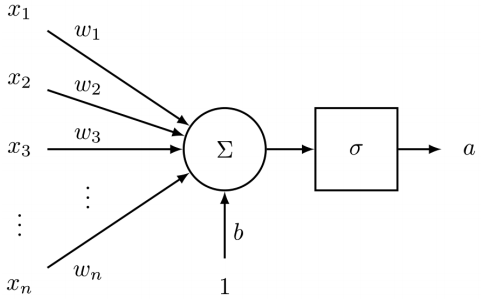
## 背景知识

前馈神经网络（Feedforward neural network，FNN），简称前馈网络，是人工神经网络的一种。前馈神经网络采用一种单向多层结构，如下图所示。图中每一个圆圈代表一个神经元，每一层包含若干个神经元。在此种神经网络中，各神经元可以接收前一层神经元的信号，并产生输出到下一层。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。



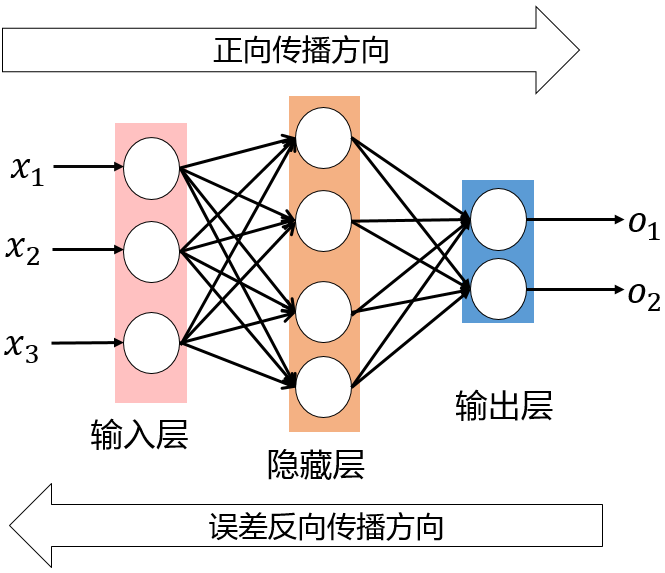
前馈网络结构图

网络中的每一个神经元都完成了一次计算工作，如下图所示。接受高维的输入(x0,x1,x2….xn)与同维度的参数(w0,w1,w2….wn)对应相乘，累加后，经过激活函数得到该神经元的输出，传递至下一层神经元。



神经元结构图

神经网络中输入数据的信息是沿着正向传播的，而预测结果和标签的误差则是通过反向传播的，在完成反向传播后就可以对所有参数进行更新。这个阶段也就是训练神经网络的过程。网络在建立时往往会随机生成所有参数。可想而知，此时的参数并不是理想参数，需要通过调整参数来达到我们想要的理想效果。这就需要训练，或者说反向传播更新参数来达到。



反向传播示例

当第一个样本输入刚刚初始化的网络，会得到一个输出值od，而对于该样本，我们知道他本该得到输出值td，因此，通过下面的损失函数，我们可以得出这个样本的损失值。当然，这只是一种损失函数：二次代价函数。

然后，就可以利用梯度下降法对网络参数进行更新。梯度下降法的思想是让损失函数沿着负梯度的方向进行搜索，不断迭代更新参数，最终使得损失值最小化。此处，会用到求偏导数和链式法则。

# 手写体图像识别实验

## 实验介绍

### 简介

Mnist手写体图像识别实验是深度学习入门经典实验。Mnist数据集包含60,000个用于训练的示例和10,000个用于测试的示例。这些数字已经过尺寸标准化并位于图像中心，图像是固定大小(28x28像素)，其值为0到255，为简单起见，每个图像都被平展并转换为784(28\*28)个特征的一维numpy数组。

### 实验目的

学会如何搭建全连接神经网络。

掌握搭建网络过程中的关键点。

掌握分类任务的整体流程。

## 实验环境要求

推荐在华为云ModelArts实验平台完成实验，也可在本地搭建python3.7.5和MindSpore1.7环境完成实验。

## 实验总体设计

**创建实验环境：**在ModelArts平台创建Ascend+MindSpore环境。

**导入实验所需模块：**该步骤通常都是程序编辑的第一步，将实验代码所需要用到的模块包用import命令进行导入。

**导入数据集并预处理：**神经网络的训练离不开数据，这里对数据进行导入。同时，因为全连接网络只能接收固定维度的输入数据，所以，要对数据集进行预处理，以符合网络的输入维度要求。同时，设定好每一次训练的Batch的大小，以Batch Size为单位进行输入。

**模型搭建：**利用mindspore.nn的cell模块搭建全连接网络，包含输入层，隐藏层，输出层。同时，配置好网络需要的优化器，损失函数和评价指标。传入数据，并开始训练模型。

**模型评估：**利用测试集进行模型的评估。

## 实验过程

### 创建实验环境

在开始本实验前，需要完成实验环境搭建工作。

进入ModelArts开发环境

参考文末附录，创建ModelArts上的开发环境Notebook并进入。

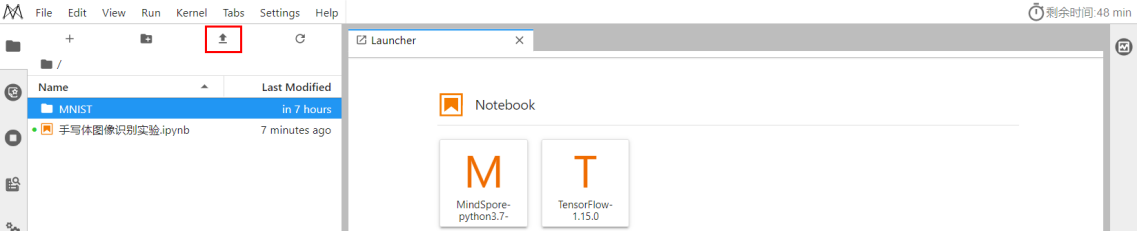
上传数据集

实验中需要用到MNIST手写体字符的数据集：<http://yann.lecun.com/exdb/mnist/>

创建MNIST文件夹，在MNIST文件夹下设置train和test文件夹，将下载的训练集和测试集数据**解压后**分别置于train和test文件夹下：

* 训练集train：train-images-idx3-ubyte，train-labels-idx1-ubyte
* 测试集test：t10k-images-idx3-ubyte，t10k-labels-idx1-ubyte

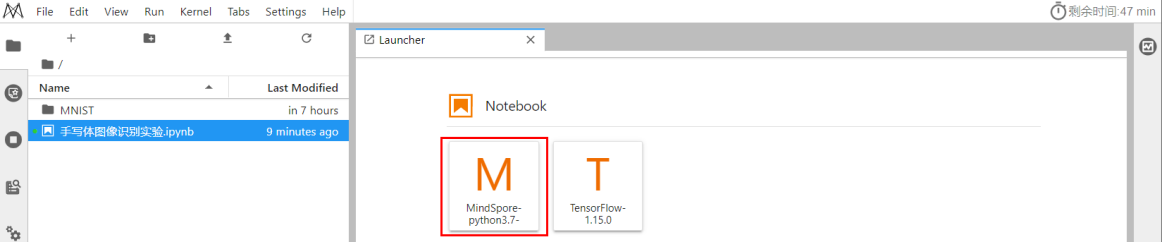
如下图所示：



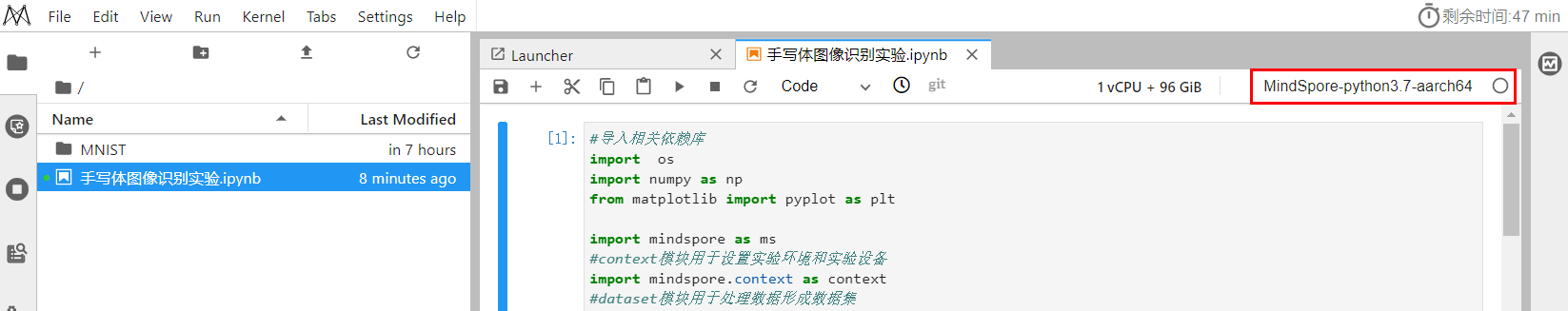
数据上传

打开Notebook

打开Notebook控制台后，新建或打开ipynb文件，选择MindSpore环境作为Kernel，即可开始编辑实验代码。



创建ipynb文件



打开ipynb文件

### 导入实验所需模块

os模块主要用于对系统路径和文件进行处理。Numpy模块主要用于数据的基本运算操作。Matplotlib模块主要用于画图。MindSpore相关模块主要用于搭建网络、调用优化器、读取数据集和将数据集处理成网络的标准输入格式。

# 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')

### 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。

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'])

输出结果：

训练数据集数量： 60000

测试数据集数量： 10000

<class 'dict'>

图像长/宽/通道数： (28, 28, 1)

一张图像的标签样式： 7

生成测试集和训练集

创建数据集，为训练集设定Batch Size，这是因为我们通常会采用小批量梯度下降法（MBGD）来训练网络，所以Batch Size作为一个非常重要的超参数需要提前设定好。在本代码中，Batch Size为128，意味着每一次更新参数，我们都用128个样本的平均损失值来进行更新。

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

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

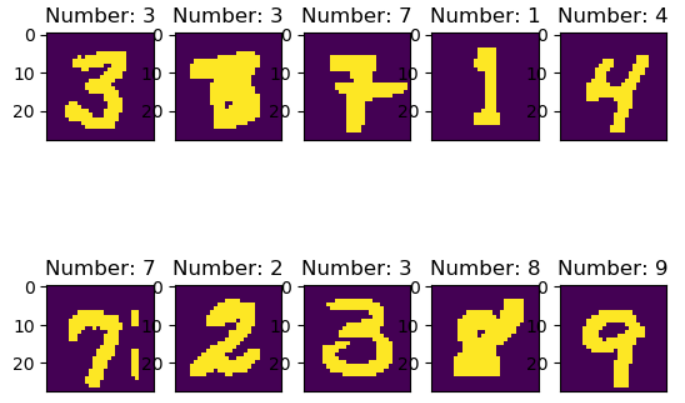
plt.imshow(np.squeeze(images[i]))

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

plt.xticks([])

plt.show()

输出结果：



输出图片结果

### 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。

创建网络

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。nn.cell能够用来组成网络模型;模型包括5个全连接层和RELU激活函数，一个全连接输出层并使用softmax进行多分类，共分成（0-9）10类

# 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

设定参数

指定模型所需的损失函数、评估指标、优化器等参数。

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)

训练模型并保存

将创建好的网络、损失函数、评估指标、优化器等参数装入模型中对模型进行训练。

# 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

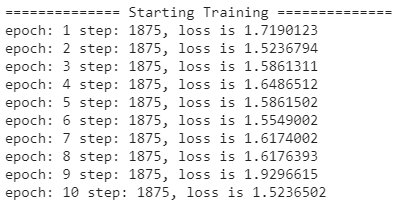
ds\_train = create\_dataset(True, batch\_size=32)

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)

输出结果：



### 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)

输出结果：



### 停止实验环境

试验完成之后请及时停止实验环境，避免资源浪费，如下图：



停止实验环境

## 实验总结

本章提供了一个基于华为MindSpore框架的手写体图像识别实验。展示了如何构建全连接网络实现图像识别的全流程。

# FashionMnist图像分类实验

## 实验介绍

### 简介

该实验主要内容是进行fashion mnist数据集标签的识别，是一个替代MNIST手写数字集的图像数据集。其涵盖了来自10种类别的共7万个不同商品的正面图片。Fashion-MNIST的大小、格式和训练集/测试集划分与原始的MNIST完全一致。60000/10000的训练测试数据划分，28x28x1的灰度图片。

### 实验目的

掌握如何使用MindSpore进行简单前馈神经网络的开发。

了解如何使用MindSpore进行简单图片分类任务的训练。

了解如何使用MindSpore进行简单图片分类任务的测试和预测。

## 实验环境要求

ModelArts平台：MindSpore

## 实验总体设计

## 实验过程

### 数据集准备

Fashion-MNIST是一个替代MNIST手写数字集的图像数据集。 它是由Zalando（一家德国的时尚科技公司）旗下的研究部门提供。其涵盖了来自10种类别的共7万个不同商品的正面图片。Fashion-MNIST的大小、格式和训练集/测试集划分与原始的MNIST完全一致。60000/10000的训练测试数据划分，28x28x1的灰度图片。

这里介绍一下经典的MNIST（手写字母）数据集。经典的MNIST数据集包含了大量的手写数字。十几年来，来自机器学习、机器视觉、人工智能、深度学习领域的研究员们把这个数据集作为衡量算法的基准之一。实际上，MNIST数据集已经成为算法作者的必测的数据集之一，但是MNIST数据集太简单了。很多深度学习算法在测试集上的准确率已经达到99.6%。

从[Fashion-MNIST GitHub仓库](https://github.com/zalandoresearch/fashion-mnist/tree/master/data/fashion" \t "_blank)下载数据集， 创建Fashion-MNIST文件夹，在Fashion-MNIST文件夹下设置train和test文件夹，将下载的训练集和测试集数据**解压后**分别置于train和test文件夹下：

训练集train：train-images-idx3-ubyte，train-labels-idx1-ubyte；

测试集test：t10k-images-idx3-ubyte，t10k-labels-idx1-ubyte；

### 创建实验环境

在开始本实验前，需要完成实验环境搭建工作。

进入ModelArts开发环境

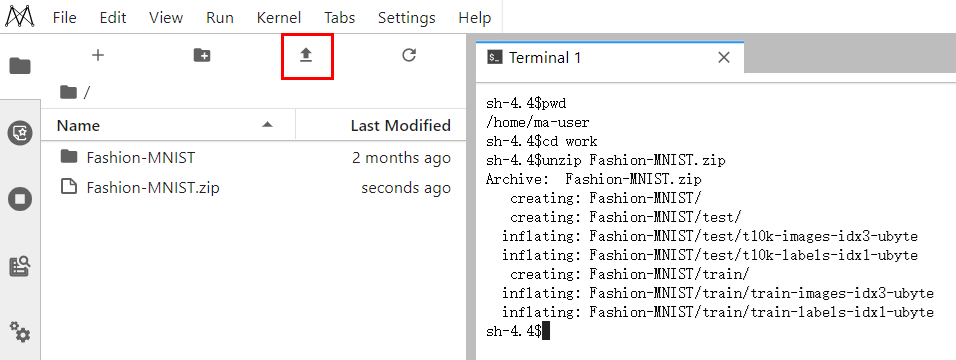
参考文末附录，创建ModelArts上的开发环境Notebook并进入。

上传数据集

将Fashion-MNIST数据集的压缩包文件（zip格式）上传至环境项目内，新建终端解压缩，输入如下命令进行解压缩，如下图所示：

cd work

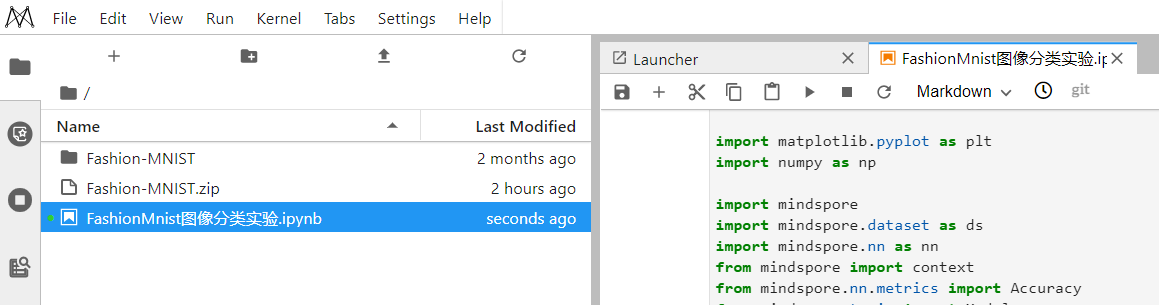
unzip Fashion-MNIST.zip



数据上传，解压缩

打开Notebook

打开Notebook控制台后，新建或打开ipynb文件，选择MindSpore环境作为Kernel，即可开始编辑实验代码。



打开ipynb文件

### 导入实验所需模块

用到的框架主要包括：

mindspore，用于神经网络的搭建；

numpy，用于处理一些数据；

matplotlib，用于画图、图像展示；

struct，用于处理二进制文件。

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')

### 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

})

### 读取并处理数据

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

数据预处理和处理结果图片展示

查看当前目录：

print(os.getcwd())

输出结果：

/home/ma-user/work

数据预处理和图片展示：

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()

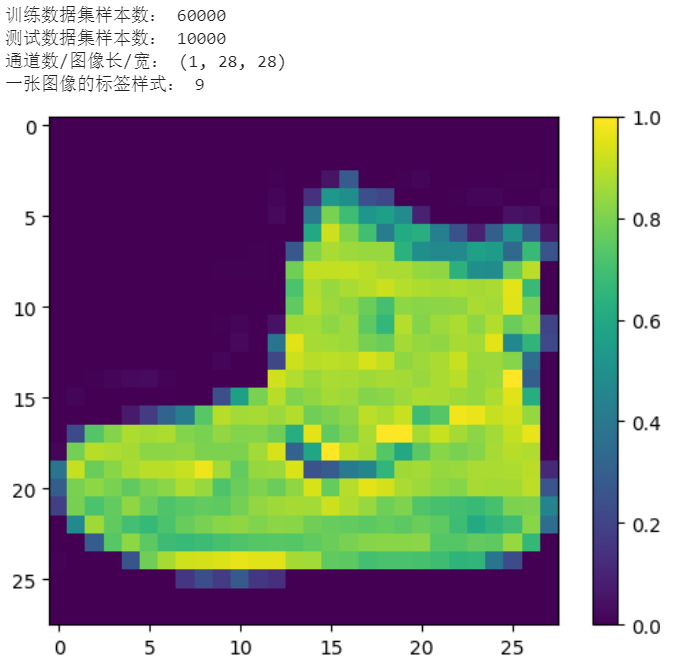
plt.imshow(train\_x[0,0,...])

plt.colorbar()

plt.grid(False)

plt.show()

输出结果：



输出结果

使用mindspore.dataset.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)

### 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)

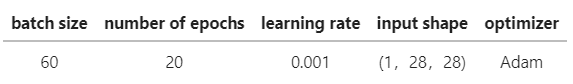
x = self.relu(x)

x = self.fc2(x)

return x

### 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)

输出结果：

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

epoch: 1 step: 1000, loss is 0.47615242

epoch: 2 step: 1000, loss is 0.6278242

epoch: 3 step: 1000, loss is 0.4200313

epoch: 4 step: 1000, loss is 0.342624

epoch: 5 step: 1000, loss is 0.20733497

epoch: 6 step: 1000, loss is 0.37861693

epoch: 7 step: 1000, loss is 0.35047245

epoch: 8 step: 1000, loss is 0.27327073

epoch: 9 step: 1000, loss is 0.23978877

epoch: 10 step: 1000, loss is 0.31236422

epoch: 11 step: 1000, loss is 0.19281301

epoch: 12 step: 1000, loss is 0.2834559

epoch: 13 step: 1000, loss is 0.1653405

epoch: 14 step: 1000, loss is 0.27971604

epoch: 15 step: 1000, loss is 0.24855426

epoch: 16 step: 1000, loss is 0.23632649

epoch: 17 step: 1000, loss is 0.18741903

epoch: 18 step: 1000, loss is 0.13822958

epoch: 19 step: 1000, loss is 0.095432095

epoch: 20 step: 1000, loss is 0.38744873

### 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)

输出结果：

{'acc': 0.8877510040160642}

实例预测：

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]])

输出结果：

第0个sample预测结果： Coat 真实结果： Coat

第1个sample预测结果： Shirt 真实结果： Shirt

第2个sample预测结果： Bag 真实结果： Bag

第3个sample预测结果： Trouser 真实结果： Trouser

第4个sample预测结果： Ankle boot 真实结果： Ankle boot

第5个sample预测结果： Dress 真实结果： Dress

第6个sample预测结果： Sneaker 真实结果： Sneaker

第7个sample预测结果： Bag 真实结果： Bag

第8个sample预测结果： Coat 真实结果： Coat

第9个sample预测结果： Dress 真实结果： Dress

第10个sample预测结果： Sneaker 真实结果： Sneaker

第11个sample预测结果： T-shirt/top 真实结果： T-shirt/top

第12个sample预测结果： Trouser 真实结果： Trouser

第13个sample预测结果： Sneaker 真实结果： Sneaker

第14个sample预测结果： Sneaker 真实结果： Sneaker

### 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()

输出结果：



输出结果

### 停止实验环境

试验完成之后请及时停止实验环境，避免资源浪费，如下图：



停止实验环境

## 实验总结

该实验主要是对全连接神经网络学习的巩固，在学习了第一个实验后，利用本实验对设计神经网络训练到评估的整体流程进行巩固。

# 汽车里程数回归预测实验

## 实验介绍

### 简介

本实验主要内容是进行汽车油耗里程数的预测，用到的框架主要包括：MindSpore，主要用于深度学习算法的构建，本实验以开源的auto-mpg数据集为基础，基于MindSpore深度学习库应用全连接神经网络进行汽车里程数预测。

### 实验目的

本实验是完成了一项回归任务。所以，本实验的重点在于区别分类任务与回归任务中网络结构的不同。

## 实验环境要求

ModelArts平台：MindSpore

## 实验总体设计

**导入实验所需模块：**该步骤通常都是程序编辑的第一步，将实验代码所需要用到的模块包用import命令进行导入。

**导入数据集并预处理：**神经网络的训练离不开数据，这里对数据进行导入。对数据集中的字符特征进行one-hot编码。同时，查看数据特征间的关联关系。

**模型搭建与训练：**利用mindspore.nn的cell模块搭建全连接网络，包含输入层，隐藏层，输出层。同时，配置好网络需要的优化器，损失函数和评价指标。传入数据，并开始训练模型。本实验是做回归任务，所以输出层的输出结果是1维的。

**查看模型训练状态：**利用MAE和MSE两个评价指标来查看模型在训练过程中的状态。

## 实验过程

环境搭建部分请参考文本附录的内容进行搭建。

数据集下载链接：<https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/deep-learning/auto-mpg.data>

### 导入实验所需模块

# 导入相关依赖库

import os

import csv

import time

import numpy as np

import pandas as pd

from matplotlib import pyplot as plt

import mindspore as ms

import mindspore.dataset as ds

import mindspore.context as context

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

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

from mindspore import nn, Tensor

from mindspore.train import Model

from mindspore.nn.metrics import Accuracy, MAE, MSE

from mindspore.train.serialization import load\_checkpoint, load\_param\_into\_net

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

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

### 导入数据集并预处理

上传数据集

参考文末附录中的步骤二将数据集文件auto-mpg.data上传至实验环境。

查看数据格式

# 加载数据集

with open('auto-mpg.data') as csv\_file:

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

print(data[20:40]) # 打印部分数据

输出结果

[['25.0 4 110.0 87.00 2672. 17.5 70 2\t"peugeot 504"'], ['24.0 4 107.0 90.00 2430. 14.5 70 2\t"audi 100 ls"'], ['25.0 4 104.0 95.00 2375. 17.5 70 2\t"saab 99e"'], ['26.0 4 121.0 113.0 2234. 12.5 70 2\t"bmw 2002"'], ['21.0 6 199.0 90.00 2648. 15.0 70 1\t"amc gremlin"'], ['10.0 8 360.0 215.0 4615. 14.0 70 1\t"ford f250"'], ['10.0 8 307.0 200.0 4376. 15.0 70 1\t"chevy c20"'], ['11.0 8 318.0 210.0 4382. 13.5 70 1\t"dodge d200"'], ['9.0 8 304.0 193.0 4732. 18.5 70 1\t"hi 1200d"'], ['27.0 4 97.00 88.00 2130. 14.5 71 3\t"datsun pl510"'], ['28.0 4 140.0 90.00 2264. 15.5 71 1\t"chevrolet vega 2300"'], ['25.0 4 113.0 95.00 2228. 14.0 71 3\t"toyota corona"'], ['25.0 4 98.00 ? 2046. 19.0 71 1\t"ford pinto"'], ['19.0 6 232.0 100.0 2634. 13.0 71 1\t"amc gremlin"'], ['16.0 6 225.0 105.0 3439. 15.5 71 1\t"plymouth satellite custom"'], ['17.0 6 250.0 100.0 3329. 15.5 71 1\t"chevrolet chevelle malibu"'], ['19.0 6 250.0 88.00 3302. 15.5 71 1\t"ford torino 500"'], ['18.0 6 232.0 100.0 3288. 15.5 71 1\t"amc matador"'], ['14.0 8 350.0 165.0 4209. 12.0 71 1\t"chevrolet impala"'], ['14.0 8 400.0 175.0 4464. 11.5 71 1\t"pontiac catalina brougham"']]

利用pandas模块读取数据

Pandas模块是一个处理表格类数据非常有效的模块。

# 使用pandas读取数据

column\_names = ['MPG','Cylinders','Displacement','Horsepower','Weight',

'Acceleration', 'Model Year', 'Origin']

# 遇到？换成nan，忽略\t之后的内容，以空格作为分隔符。

raw\_data = pd.read\_csv('auto-mpg.data', names=column\_names,

na\_values = "?", comment='\t',

sep=" ", skipinitialspace=True)

data = raw\_data.copy()

# 查看数据形状

data.shape

输出结果

(398, 8)

数据预处理

对于数据集中的空值，我们要在建模前进行处理。此处空值的数据较少，我们直接进行删除。特征Origin代表着车辆的归属区域信息，此处总共分为三种，欧洲，美国，日本，我们需要对此特征进行one-hot编码。

# 对于数据集中的空值，我们要在建模前进行处理。此处空值的数据较少，我们直接进行删除。

# 清洗空数据

data = data.dropna()

data.tail()

# Pandas库提供了简单的数据集统计信息，我们可直接调用函数describe()进行查看。

# 查看训练数据集的结构

origin = data.pop('Origin')

data\_labels = data.pop('MPG')

train\_stats = data.describe()

train\_stats = train\_stats.transpose()

train\_stats

# 归一化数据

def norm(x):

return (x - train\_stats['mean']) / train\_stats['std']

normed\_data = norm(data)

# 将MPG放回归一化后的数据中

normed\_data['MPG'] = data\_labels

# 离散特征处理

# 特征Origin代表着车辆的归属区域信息，此处总共分为三种，欧洲，美国，日本，我们需要对此特征进行one-hot编码。

# 对origin属性进行one-hot编码

normed\_data['USA'] = (origin == 1)\*1.0

normed\_data['Europe'] = (origin == 2)\*1.0

normed\_data['Japan'] = (origin == 3)\*1.0

模型训练需要区分特征值与目标值，也就是我们常说的X值与Y值，此处MPG为Y值，其余的特征为X值。

在模型构建的时候，我们一般需要准备两份数据，一份训练集数据用于模型的训练构建，一份测试集用于模型的评估优化。创建比例0.8，用于分割训练集和验证集，80%的数据用于模型训练，20%的数据用于模型验证。

# 将数据集按照4：1划分成训练集和测试集

train\_dataset = normed\_data.sample(frac=0.8,random\_state=0)

test\_dataset = normed\_data.drop(train\_dataset.index)

# 模型训练需要区分特征值与目标值，也就是我们常说的X值与Y值，此处MPG为Y值，其余的特征为X值。

# 将目标值和特征分开

train\_labels = train\_dataset.pop('MPG')

test\_labels = test\_dataset.pop('MPG')

X\_train, Y\_train = np.array(train\_dataset), np.array(train\_labels)

X\_test, Y\_test = np.array(test\_dataset), np.array(test\_labels)

# 查看数据集尺寸

print('训练数据x尺寸：',X\_train.shape)

print('训练数据y尺寸：',Y\_train.shape)

print('测试数据x尺寸：',X\_test.shape)

print('测试数据y尺寸：',Y\_test.shape)

# 将数据集转换为Tensor格式

ds\_xtrain= Tensor(X\_train, ms.float32)

ds\_ytrain= Tensor(Y\_train, ms.int32)

ds\_xtest=Tensor(X\_test, ms.float32)

ds\_ytest=Tensor(Y\_test, ms.int32)

输出结果

训练数据x尺寸： (314, 9)

训练数据y尺寸： (314,)

测试数据x尺寸： (78, 9)

测试数据y尺寸： (78,)

### 模型搭建与训练

创建网络

汽车油耗里程数据集准备完成，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。nn.cell能够用来组成网络模型;模型共包含3个全连接，采用Relu当做激活函数

# 定义网络

class Regression\_car(nn.Cell):

def \_\_init\_\_(self):

super(Regression\_car, self).\_\_init\_\_()

self.flatten = nn.Flatten()

self.relu = nn.ReLU()

self.fc1 = nn.Dense(9,64, activation='relu')

self.fc2 = nn.Dense(64,64, activation='relu')

self.fc3 = nn.Dense(64,1)

def construct(self, x):

x = self.flatten(x)

x = self.fc1(x)

x = self.fc2(x)

x = self.fc3(x)

return x

设定参数并训练模型

指定模型所需的损失函数、评估指标、优化器等参数。回归问题，损失函数采用均方误差MSE。将创建好的网络、损失函数、评估指标、优化器等参数装入模型中对模型进行训练。需要逐步打印出MAE、MSE的值。

# 定义网络，损失函数，评估指标，优化器

network = Regression\_car()

net\_loss = nn.MSELoss()

net\_opt = nn.RMSProp(network.trainable\_params(), 0.001)

# 使用单步训练的方式来使结果中打印出MAE、MSE

# Cell是所有神经网络的基类。Cell可以是一个单一的神经网络单元，如conv2d、relu、batch\_norm等，也可以是一个用于构建网络的单元的组合。

# WithLossCell意味着使用损耗函数对网络进行包裹。此单元接受数据和标签作为输入，并返回计算的损耗。

with\_loss=nn.WithLossCell(network, net\_loss)

train\_step = nn.TrainOneStepCell(with\_loss, net\_opt).set\_train()

# WithEvalCell返回loss、输出和标签的单元，用于评估。此单元接受网络和loss函数作为参数，并计算模型的loss。它返回loss、输出和标签来计算度量。

evalcell=nn.WithEvalCell(network,net\_loss)

# 创建指标类

mae = nn.MAE()

mse = nn.MSE()

val\_mae = nn.MAE()

val\_mse = nn.MSE()

# 创建一个空的Dataframe

result =pd.DataFrame(columns=('\_epoch','\_loss','\_mae','\_mse','val\_mae','val\_mse'))

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

for epoch in range(300):

# 利用train\_step去接收训练集，更新网络参数得到loss值

loss = train\_step(ds\_xtrain,ds\_ytrain)

# 利用evalcell接收训练集获取训练过程的输出用于计算mae和mse，接收测试集获取测试集输出

\_, outputs, label = evalcell(ds\_xtrain,ds\_ytrain)

\_, val\_outputs, val\_label = evalcell(ds\_xtest,ds\_ytest)

# 每次循环都更新MAE、MSE等的值。

mae.clear()

mae.update(outputs, label)

mse.clear()

mse.update(outputs, label)

val\_mae.clear()

val\_mae.update(val\_outputs, val\_label)

val\_mse.clear()

val\_mse.update(val\_outputs, val\_label)

Mae = mae.eval()

Mse = mse.eval()

Val\_Mae = val\_mae.eval()

Val\_Mse = val\_mse.eval()

nd\_loss = loss.asnumpy()

fl\_loss = float(nd\_loss)/24.0

# 将计算结果逐行插入result,注意变量要用[]括起来,同时ignore\_index=True，否则会报错，ValueError: If using all scalar values, you must pass an index

result=result.append(pd.DataFrame({'\_epoch':[epoch],'\_loss':fl\_loss,'\_mae':Mae,'\_mse':Mse,

'val\_mae':Val\_Mae,'val\_mse':Val\_Mse}),ignore\_index=True)

if epoch%10==0:

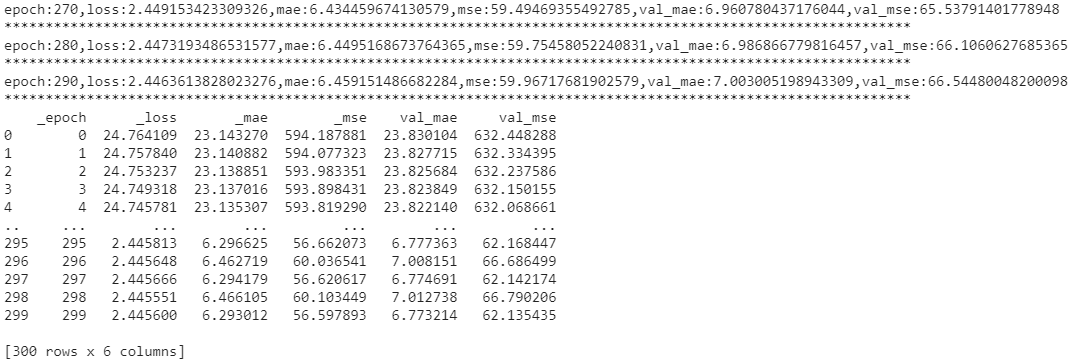
print('epoch:{0},loss:{1},mae:{2},mse:{3},val\_mae:{4},val\_mse:{5}'.format(epoch,fl\_loss,Mae,Mse,

Val\_Mae,Val\_Mse))

print("\*" \* 110)

print(result)

输出结果：



输出结果

### 查看模型训练状态

绘制模型损失函数图，利用result中的MAE和MSE查看模型训练状态。

# 绘制模型损失函数图

def plot\_history(result):

plt.figure()

plt.xlabel('Epoch')

plt.ylabel('Mean Abs Error [MPG]')

plt.plot(result['\_epoch'], result['\_mae'],

label='Train Error')

plt.plot(result['\_epoch'], result['val\_mae'],

label = 'Val Error')

plt.ylim([0,20])

plt.legend()

plt.figure()

plt.xlabel('Epoch')

plt.ylabel('Mean Square Error [$MPG^2$]')

plt.plot(result['\_epoch'], result['\_mse'],

label='Train Error')

plt.plot(result['\_epoch'], result['val\_mse'],

label = 'Val Error')

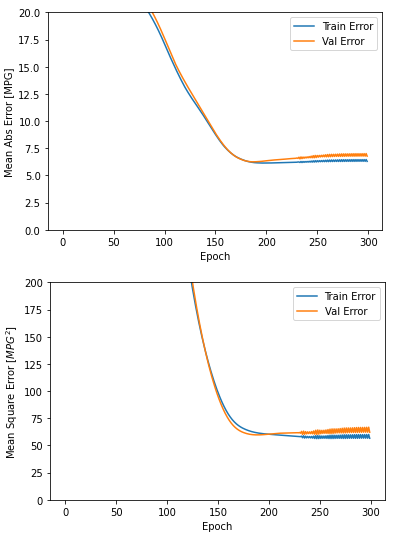
plt.ylim([0,200])

plt.legend()

plt.show()

plot\_history(result)

输出结果：



输出结果

### 停止实验环境

试验完成之后请及时停止实验环境，避免资源浪费，如下图：



停止实验环境

## 实验总结

本实验利用全连接网络完成了汽车油耗里程数预测的任务。重点理解在分类任务和回归任务中，网络模型架构的不同。最后，利用训练返回值中的相关参数，学习在训练中，模型状态的变化，从而深入理解网络训练过程。

# 鸢尾花分类任务对比实验

## 实验介绍

### 简介

优化器是神经网络训练过程中非常重要的结构，正是因为优化器的存在帮助初始化参数的网络一步步学习到了符合训练集数据特征的最优参数。本实验主要介绍如何使用MindSpore进行优化实验。共包含两个实验，分别为：

实验一：利用不同优化器求解函数极值点实验

实验二：鸢尾花数据在不同优化器下的分类实验

通过分析无优化器、SGD优化器、Momentum优化器、Adam优化器模型在求极值点和分类任务下的结果 ，得到不同优化器的作用。

### 实验目的

掌握MindSpore中优化器的使用及其作用。

了解如何使用MindSpore进行SGD、Momentum、Adam优化器实验。

了解不同优化器原理。

## 实验环境要求

ModelArts平台：MindSpore

## 实验总体设计

本实验主要分为两个部分，第一部分重点在于理解各个优化器的原理，自己编写优化器，体会优化器的作用，主要设计思路如下：

**构建Beale 公式及其导数：**Beale 公式是一个经典的二元函数，它在三维空间中有一个复杂的曲面，方程最低点在(3,0.5)处。该部分首先要定义这个函数并定义其偏导数。这部分重点掌握如何将数学公式在python中实例化。

**编写动量优化器并优化Beale 公式：**通过编写动量优化器函数来优化上一步定义好的beale函数。该步骤重点掌握优化器函数是如何实现的和动量优化器的原理。

**编写Adagrad优化器并优化Beale 公式：**这步与上一步类似，重点掌握Adagrad优化器的原理。

第二部分主要通过构建全连接网络体会有无优化器的全连接网络的差别，从而深入理解优化器在神经网络模型构建中的作用，主要设计流程如下：

导入模块和数据集、数据预处理和构建神经网络这三步与第二章、第四章的设计相同。

**无优化器模型预测**：主要是初始化网络后，不经过训练，直接进行预测任务。

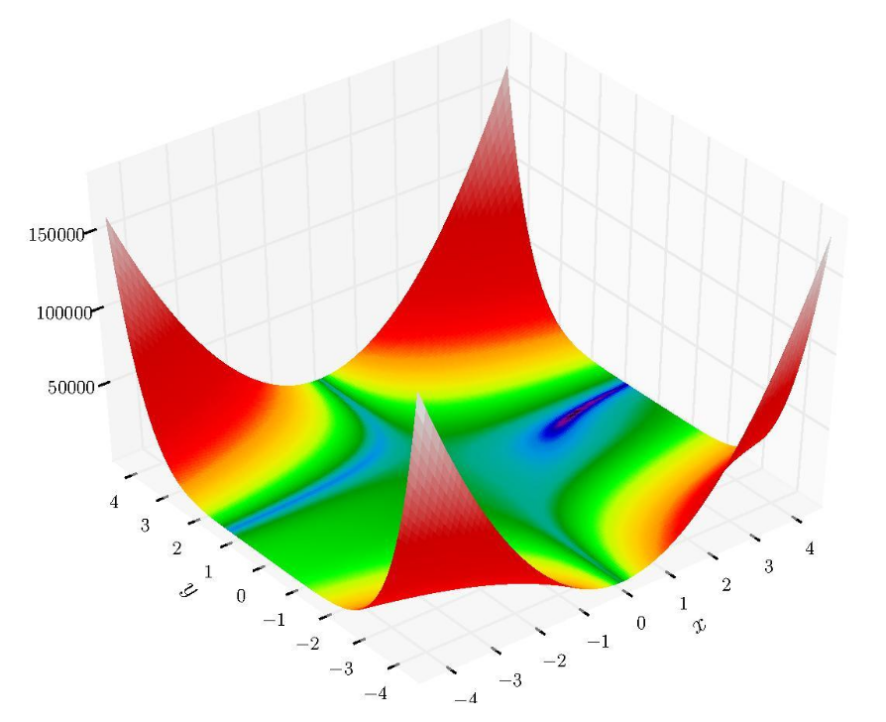
**有优化器模型训练并预测：**与通常训练完网络并进行预测的步骤相同。重点是结合上一步，体会优化器对于神经网络的作用。

## 背景知识

Beale 公式：



表面如下图：



这个方程和这个曲面就是我们要优化的对象，优化的过程，就是我们随机给定曲面上的一个点，经过多次迭代。最终，找到曲面上最低的那个点。

## 实验过程

环境搭建部分请参考文末附录的内容进行搭建。

数据集下载链接：<https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/deep-learning/iris.data>

### 构建Beale 公式及其导数

导入相关模块

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.colors as plt\_cl

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)

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)

return dfdx1, dfdx2

可视化beale方程

# 定义画图函数

def gd\_plot(x\_traj):

plt.rcParams['figure.figsize'] = [6, 6]

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

norm=plt\_cl.LogNorm(), cmap=plt.cm.jet)

plt.title('2D Contour Plot of Beale function(Momentum)')

plt.xlabel('$x\_1$')

plt.ylabel('$x\_2$')

plt.axis('equal')

plt.plot(3, 0.5, 'k\*', markersize=10)

if x\_traj is not None:

x\_traj = np.array(x\_traj)

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

plt.show()

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))

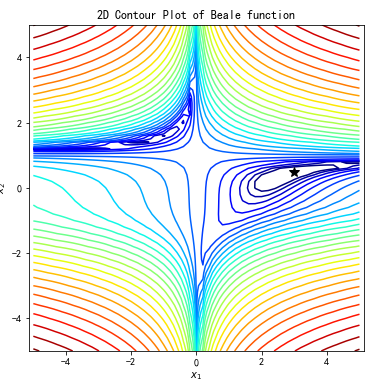
Y = beale(X1, X2)

print("目标结果 (x\_1, x\_2) = (3, 0.5)")

gd\_plot(None)

输出结果：

目标结果 (x\_1, x\_2) = (3, 0.5)



### 编写不使用优化器优化Beale 公式

无优化器训练不更新参数，不管训练多少次，模型的参数没有发生变化。模型效果与模型初始化参数关系较大。

构建无优化器函数

# 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

优化beale方程

# 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

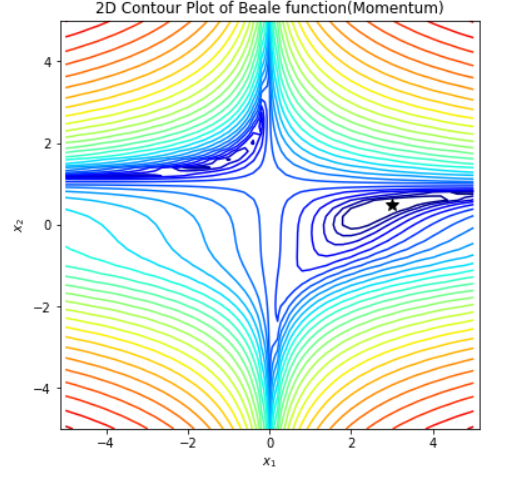
x\_traj\_no = gd\_no(dbeale\_dx, x0, conf\_para\_no)

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)

输出结果：

无优化器求得极值点 (x\_1, x\_2) = (1.0, 1.5)



### 编写SGD优化器并优化Beale 公式

梯度下降法：梯度下降（gradient descent）在机器学习中应用十分的广泛，是求解无约束优化问题最简单和最古老的方法之一。通过迭代，参数向梯度的反方向更新，直到收敛。



其中表示损失函数 J 关于参数W的梯度表示学习率；

缺点：

有可能会陷入局部最小值；

不会收敛，最终会一直在最小值附近波动，并不会达到最小值并停留在此；

下降速度慢；

选择合适的学习率比较困难；

在所有方向上统一的缩放梯度，不适用于稀疏数据；

构建SGD优化器

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]))

v = - conf\_para['learning\_rate'] \* dfdx

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

return x\_traj

优化beale方程

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

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

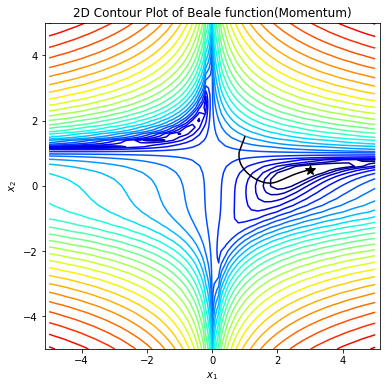
x\_traj\_sgd = gd\_sgd(dbeale\_dx, x0, conf\_para\_sgd)

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)

输出结果：

SGD求得极值点 (x\_1, x\_2) = (2.981957257903412, 0.4954609564102124)



### 编写动量优化器并优化Beale 公式

Momentum：是动量优化法中的一种（Momentum、NAG），即使用动量(Momentum)的随机梯度下降法(SGD)，主要思想是引入一个积攒历史梯度信息的动量来加速SGD。其参数优化公式如下所示：



其中表示损失函数 J 关于参数W的梯度表示学习率；表示动量的大小，一般取值为0.9。

优点：

参考了历史梯度，增加了稳定性；

由于引入加速动量，加快收敛速度。

还有一定摆脱局部最优的能力。

构建Momentum优化器

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]))

v = conf\_para['momentum'] \* v - conf\_para['learning\_rate'] \* dfdx

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

return x\_traj

优化beale方程

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

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

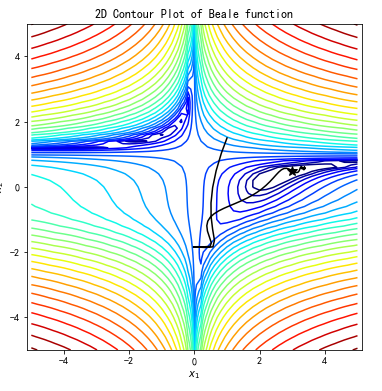
x\_traj\_momentum = gd\_momentum(dbeale\_dx, x0, conf\_para\_momentum)

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)

输出结果：

Momentum求得极值点 (x\_1, x\_2) = (3.0004236971560485, 0.5001057348328146)



### 编写自适应优化器并优化Beale 公式

自适应学习率优化算法主要有：AdaGrad算法，RMSProp算法，Adam算法以及AdaDelta算法。

AdaGrad

AdaGrad的基本思想是对每个变量用不同的学习率。这个学习率在一开始比较大，用于快速梯度下降。随着优化过程的进行，对于已经下降很多的变量，则减缓学习率，对于还没怎么下降的变量，则保持一个较大的学习率。其参数优化公式如下所示：



其中表示损失函数 J 关于参数W的梯度；表示学习率，一般取值0.01；是一个很小的数，防止分母为0；表示了前t 步参数梯度的平方累加。把沿路的Gradient的平方根，作为Regularizer。分母作为Regularizer项的工作机制如下：

训练前期，梯度较小，使得Regularizer项很大，放大梯度。[激励阶段]

训练后期，梯度较大，使得Regularizer项很小，缩小梯度。[惩罚阶段]

优点：

在数据分布稀疏的场景，能更好利用稀疏梯度的信息，比标准的SGD算法更有效地收敛；

对每个变量用不同的学习率，对输入参数学习率的依赖小，容易调节参数；

缺点：

主要缺陷来自分母项的对梯度平方不断累积，随之时间步地增加，分母项越来越大，最终导致学习率收缩到太小无法进行有效更新；

编写adagrad优化器

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

优化beale方程

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

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

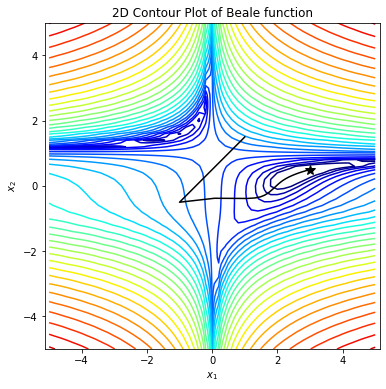
x\_traj\_adag = gd\_adagrad(dbeale\_dx, x0, conf\_para\_adag)

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)

输出结果：

Adagrad求得极值点 (x\_1, x\_2) = (2.9993173156940776, 0.49982846432011524)



### 思考题

思考题1：分析实验结果，对比各分类器不同。

思考题2：代码np.logspace的作用是什么？

### 导入模块、读取鸢尾花数据集并预处理、构建神经网络

该案例主要内容是进行鸢尾花数据集标签的识别，鸢尾花数据集共收集了三类鸢尾花，即Setosa鸢尾花、Versicolour鸢尾花和Virginica鸢尾花，每一类鸢尾花收集了50条样本记录，共计150条。可参考文末附录中的步骤二将数据集文件iris.data上传至实验环境。

数据集包括4个属性，分别为花萼的长、花萼的宽、花瓣的长和花瓣的宽。对花瓣我们可能比较熟悉，花萼是什么呢？花萼是花冠外面的绿色被叶，在花尚未开放时，保护着花蕾。四个属性的单位都是cm，属于数值变量，1个标签，表示鸢尾花的分类。

导入模块和定义变量

用到的框架主要包括：MindSpore，专用于搭建整体的网络框架，这里主要用于神经网络的搭建。另外 os也被引入，它提供了丰富的函数，主要用来处理文件和目录。

实验代码：

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

# 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,

'data\_dir': './iris.data',

'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

})

读取数据并预处理

# 读取鸢尾花数据集，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=','))

# 共150条数据，将数据集的4个属性作为自变量X。将数据集的3个类别映射为{0, 1，2}，作为因变量Y

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]

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

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'])

# 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)

# 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

定义训练、评估、测试函数

# 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")

# 定义模型

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

# 定义损失值指标

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

# 设置checkpoint

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

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

ckpoint\_cb = ModelCheckpoint(prefix=prefix, directory=directory, config=config\_ck)

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

# 训练模型

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

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\_\_()

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

predictions = model.predict(test)

predictions = predictions.asnumpy()

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

for i in range(10):

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]])

### 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

输出结果：

------------------无优化器--------------------------

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

epoch: 1 step: 4, loss is 1.1123106

epoch: 2 step: 4, loss is 1.1202087

epoch: 3 step: 4, loss is 1.1081936

epoch: 4 step: 4, loss is 1.1139123

epoch: 5 step: 4, loss is 1.1177686

epoch: 6 step: 4, loss is 1.1094363

epoch: 7 step: 4, loss is 1.1057143

epoch: 8 step: 4, loss is 1.1137706

epoch: 9 step: 4, loss is 1.1225764

epoch: 10 step: 4, loss is 1.1169411

epoch: 11 step: 4, loss is 1.115051

epoch: 12 step: 4, loss is 1.1098907

epoch: 13 step: 4, loss is 1.1069646

epoch: 14 step: 4, loss is 1.1149268

epoch: 15 step: 4, loss is 1.1180775

epoch: 16 step: 4, loss is 1.1170397

epoch: 17 step: 4, loss is 1.1150074

epoch: 18 step: 4, loss is 1.1237894

epoch: 19 step: 4, loss is 1.1233115

epoch: 20 step: 4, loss is 1.116143

{'acc': 0.43333333333333335}

第0个sample预测结果： versicolor 真实结果： virginica

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

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

第3个sample预测结果： versicolor 真实结果： setosa

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

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

第6个sample预测结果： setosa 真实结果： virginica

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

第8个sample预测结果： versicolor 真实结果： setosa

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

### SGD优化器模型训练并预测

# ---------------------------------------------------SGD-------------------------------------

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

输出结果：

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

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

epoch: 1 step: 4, loss is 1.0863036

epoch: 2 step: 4, loss is 1.0471957

epoch: 3 step: 4, loss is 1.0281278

epoch: 4 step: 4, loss is 1.0003911

epoch: 5 step: 4, loss is 1.0010878

epoch: 6 step: 4, loss is 0.9402846

epoch: 7 step: 4, loss is 0.936496

epoch: 8 step: 4, loss is 0.9070135

epoch: 9 step: 4, loss is 0.8746185

epoch: 10 step: 4, loss is 0.8588983

epoch: 11 step: 4, loss is 0.8791269

epoch: 12 step: 4, loss is 0.8409094

epoch: 13 step: 4, loss is 0.8171078

epoch: 14 step: 4, loss is 0.78627276

...

{'acc': 1.0}

第0个sample预测结果： versicolor 真实结果： versicolor

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

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： setosa 真实结果： setosa

第8个sample预测结果： virginica 真实结果： virginica

第9个sample预测结果： setosa 真实结果： setosa

### 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

输出结果：

-------------------Momentum优化器-----------------------

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

epoch: 1 step: 4, loss is 1.0810095

epoch: 2 step: 4, loss is 0.94718075

epoch: 3 step: 4, loss is 0.8274425

epoch: 4 step: 4, loss is 0.66691095

epoch: 5 step: 4, loss is 0.6318491

epoch: 6 step: 4, loss is 0.57523024

epoch: 7 step: 4, loss is 0.56330514

epoch: 8 step: 4, loss is 0.481749

epoch: 9 step: 4, loss is 0.39138663

epoch: 10 step: 4, loss is 0.4436511

epoch: 11 step: 4, loss is 0.4150259

epoch: 12 step: 4, loss is 0.4283092

epoch: 13 step: 4, loss is 0.41189855

epoch: 14 step: 4, loss is 0.38341784

epoch: 15 step: 4, loss is 0.38927388

epoch: 16 step: 4, loss is 0.39323956

epoch: 17 step: 4, loss is 0.38287503

epoch: 18 step: 4, loss is 0.43862677

epoch: 19 step: 4, loss is 0.3592914

epoch: 20 step: 4, loss is 0.35841298

{'acc': 0.9666666666666667}

第0个sample预测结果： versicolor 真实结果： versicolor

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

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

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

第6个sample预测结果： setosa 真实结果： setosa

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

第8个sample预测结果： virginica 真实结果： virginica

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

### 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)

输出结果：

-----------------Adam优化器--------------------------

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

epoch: 1 step: 4, loss is 0.85976017

epoch: 2 step: 4, loss is 0.6710789

epoch: 3 step: 4, loss is 0.5583926

epoch: 4 step: 4, loss is 0.5004689

epoch: 5 step: 4, loss is 0.37697986

epoch: 6 step: 4, loss is 0.39567876

epoch: 7 step: 4, loss is 0.3641772

epoch: 8 step: 4, loss is 0.31769836

epoch: 9 step: 4, loss is 0.33615974

epoch: 10 step: 4, loss is 0.36870608

epoch: 11 step: 4, loss is 0.31535652

epoch: 12 step: 4, loss is 0.29351452

epoch: 13 step: 4, loss is 0.27576455

epoch: 14 step: 4, loss is 0.24675885

epoch: 15 step: 4, loss is 0.31923586

{'acc': 1.0}

{'acc': 0.9666666666666667}

第0个sample预测结果： setosa 真实结果： setosa

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

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

第3个sample预测结果： versicolor 真实结果： versicolor

第4个sample预测结果： setosa 真实结果： setosa

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

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： setosa 真实结果： setosa

### 查看保存的模型

os.listdir('./model\_iris/adam') # 查看保存的模型

输出结果：

['checkpoint\_adam-graph.meta', 'checkpoint\_adam-15\_4.ckpt']

### 停止实验环境

试验完成之后请及时停止实验环境，避免资源浪费，如下图：



停止实验环境

## 结果分析

从无优化器、SGD、momentum、adam优化器的loss上看：

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，学员可以自己尝试。

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，学员可以自己尝试。

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。

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。

## 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：

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；

有优化器条件下训练，更新模型参数。训练loss改变；

SGD优化器收敛速度慢，参数不容易调节；

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；

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；

# 附录：ModelArts开发环境搭建

在华为云ModelArts平台上创建AI框架为Mindspore-1.7，硬件环境为Ascend 910+ARM的开发环境。

进入华为云ModelArts控制台

在[华为云ModelArts主页](https://www.huaweicloud.com/product/modelarts.html)，点击“管理控制台”进入ModelArts的管理页面。

图形用户界面, 文本, 应用程序

描述已自动生成

华为云ModelArts主页

创建Notebook训练作业

控制台区域选择“华北-北京四”，在左侧菜单栏中选择“开发环境”的“Notebook”，点击进入Notebook创建页面。



ModelArts控制台

点击“创建”按钮，创建一个新的Notebook，其配置如下：

名称：自定义。

自动停止：建议1小时。

镜像：Ascend+ARM算法开发和训练基础镜像。

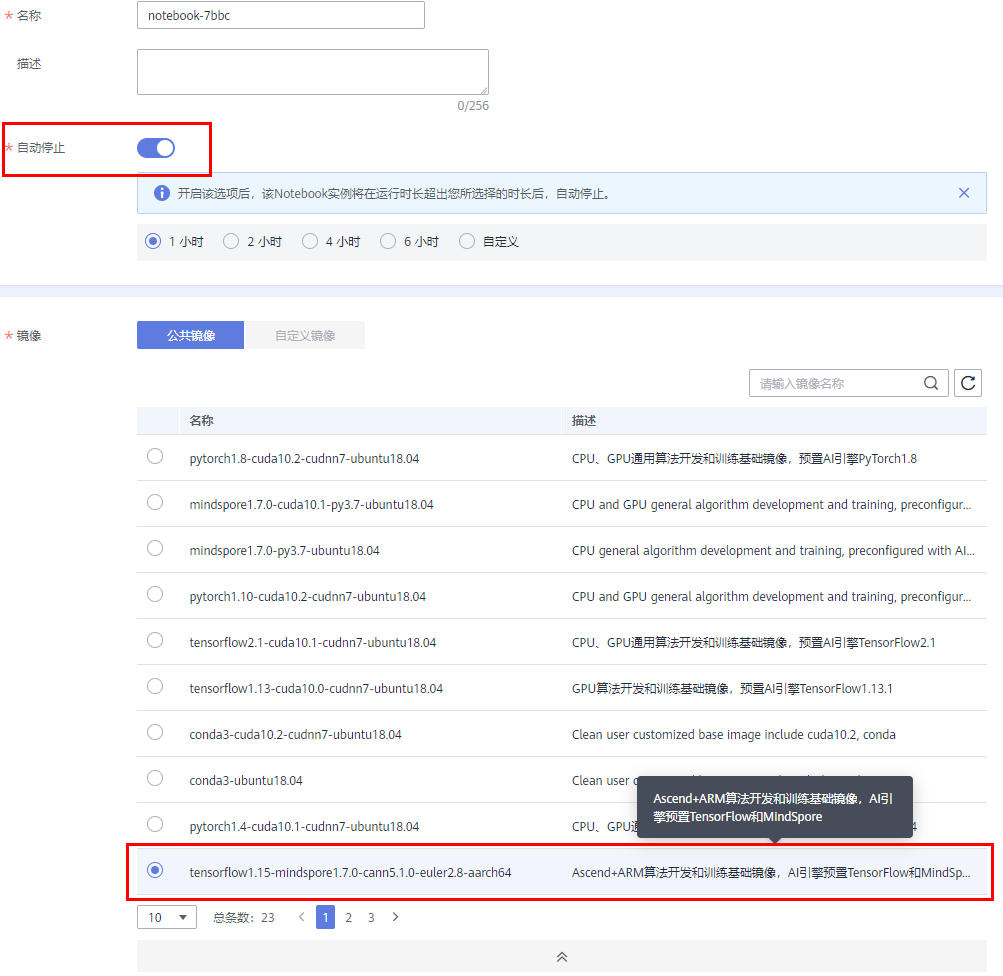
资源池：公共资源池。

类型：ASCEND。

规格：Ascend: 1\*Ascend910|CPU: 24核 96GB。

存储配置：默认存储（50GB），亦可选择EVS，支持自定义存储规格且为专属资源。

如图所示：





Notebook创建配置

配置完成之后“立即创建”，规格确认无误之后“提交”。

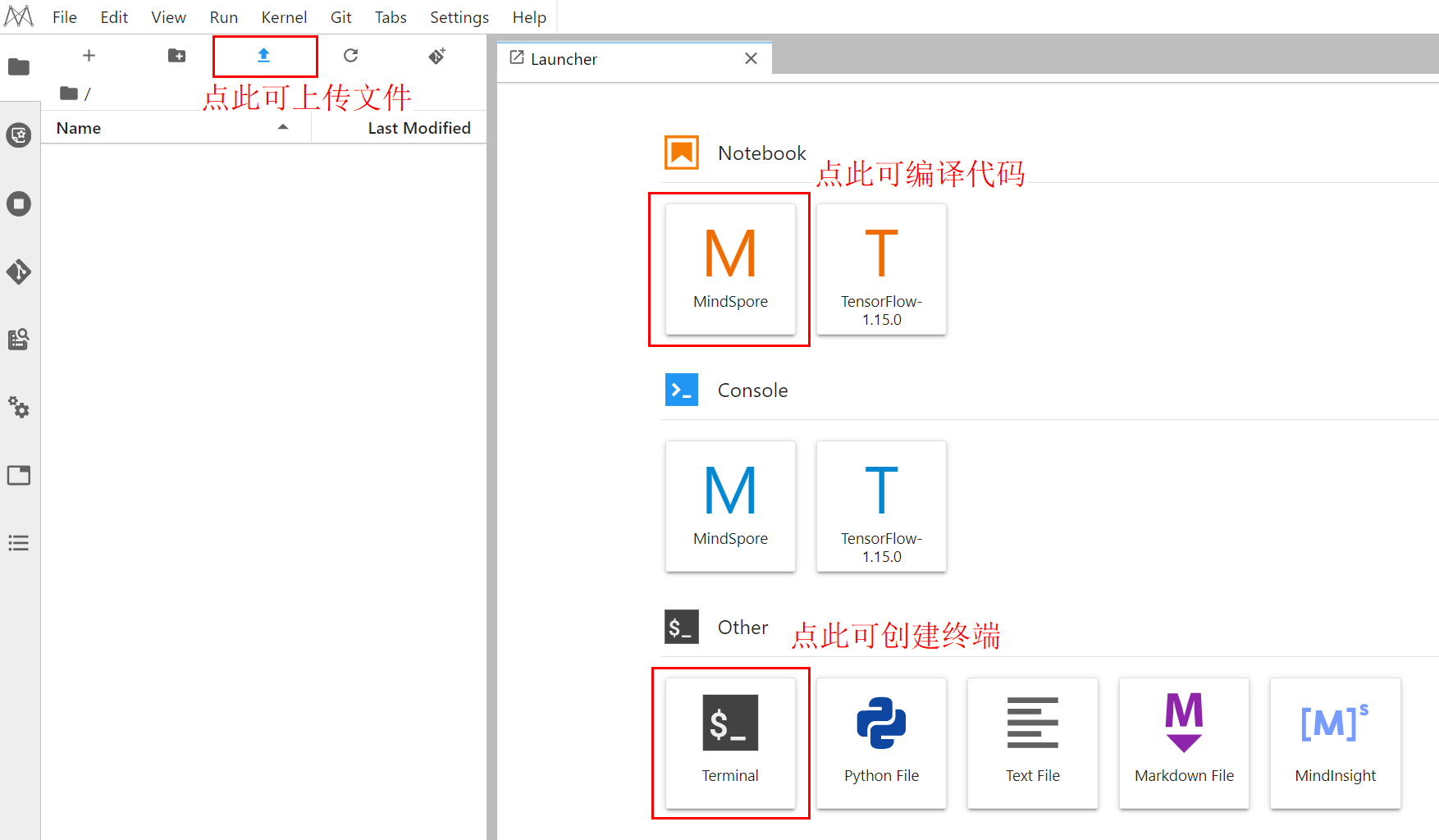
启动Notebook进入开发环境

当上一步创建好Notebook状态显示为“运行中”时，在右侧“操作”栏中“打开”，即可进入在线编程页面。



打开Notebook环境

可以在此页面创建或编辑MindSpore的项目，如图所示：



Notebook开发页面

\*注意：Notebook环境内上传、创建和编辑的文件均在/home/ma-user/work目录下。

停止Notebook训练作业

实验完成之后，请及时关闭Notebook训练作业，避免产生不必要的资源浪费。

登录[华为云ModelArts控制台](https://console.huaweicloud.com/modelarts/?region=cn-north-4" \l "/dev-container)，在“操作”栏选择“停止”操作。

如下图所示：



及时停止Notebook

至此训练用的线上Notebook环境搭建完成。